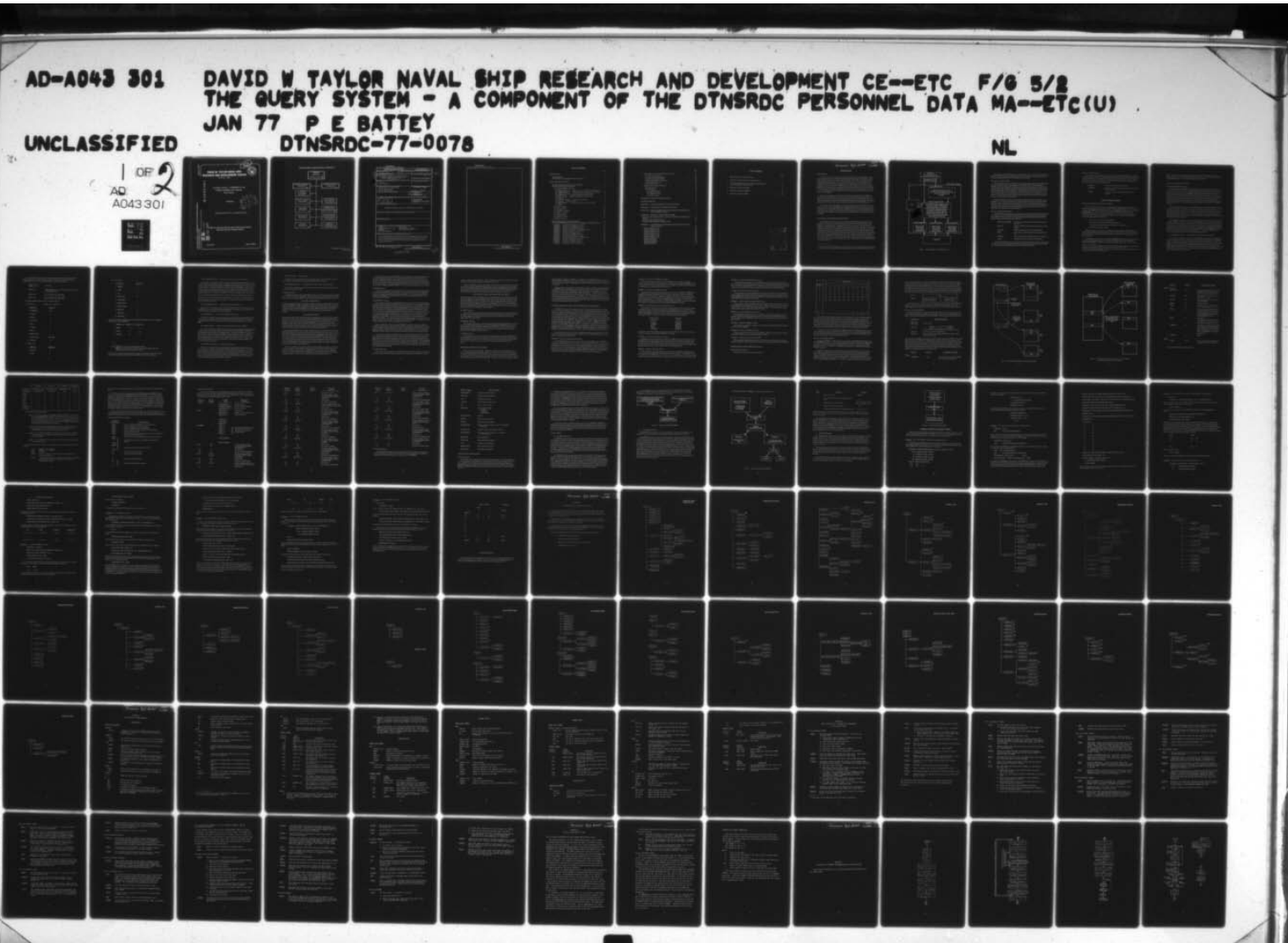
AD-A043 301 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 5/2
THE QUERY SYSTEM - A COMPONENT OF THE DTNSRDC PERSONNEL DATA MA--ETC(U)
JAN 77 P E BATTEY

UNCLASSIFIED DTNSRDC-77-0078 NL

1 OF 2
AD
A043301



Report 77-0078

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Md. 20084

THE QUERY SYSTEM – A COMPONENT OF THE DTNSRDC PERSONNEL DATA MANAGEMENT SYSTEM

Philip Battey

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

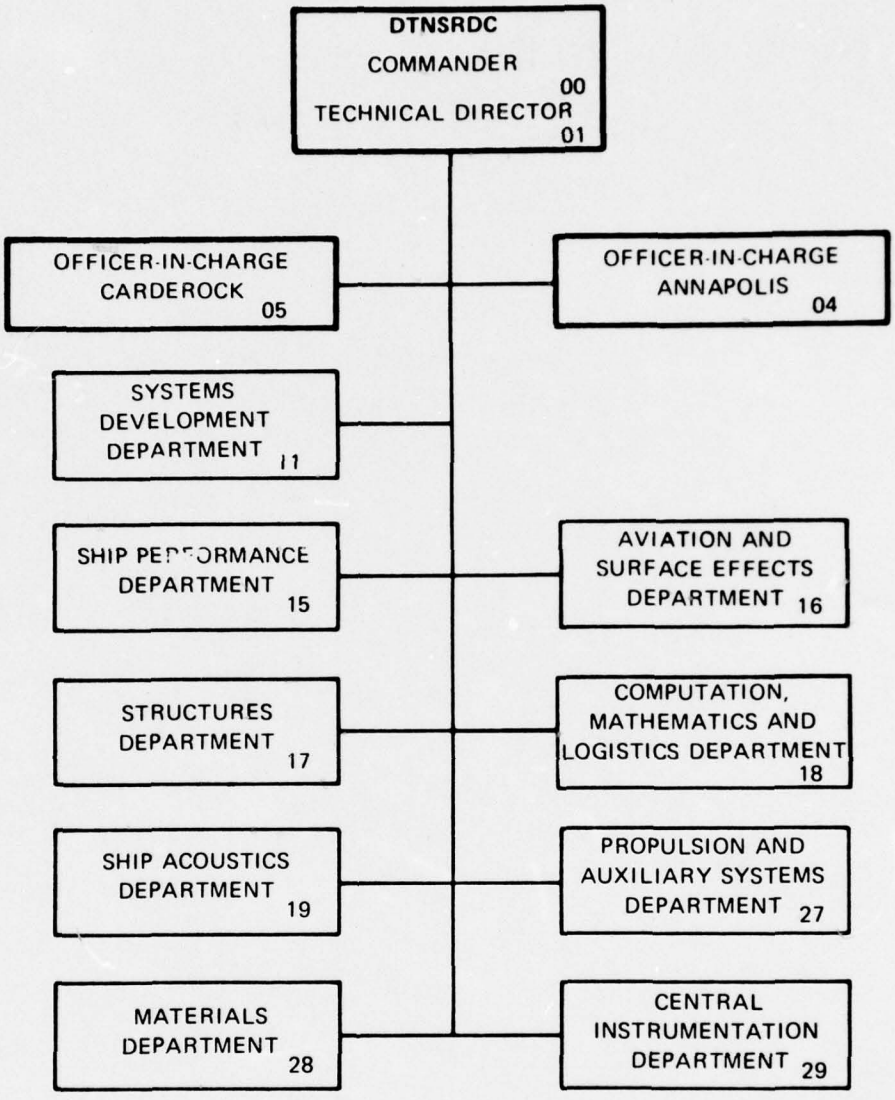
COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

January 1977

Report 77-0078

THE QUERY SYSTEM – A COMPONENT OF THE DTNSRDC PERSONNEL DATA MANAGEMENT SYSTEM

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



GPO 905-678

NDW-DTNSRDC 3960/43b (Rev. 11-75)
GPO 928-108

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DTNSRDC-77-0078	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): THE QUERY SYSTEM - A COMPONENT OF THE DTNSRDC PERSONNEL DATA MANAGEMENT SYSTEM,	5. TYPE OF REPORT & PERIOD COVERED: Research and development rept.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Philip E. Battey	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: David W. Taylor Naval Ship Research and Development Center, Bethesda, Maryland 20084	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit 5-7100-018-49
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: January 1977
	13. NUMBER OF PAGES: 188
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 187p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CAPMIS
COMPUTERIZED DATA BASE
COMRADE
DATA MANAGEMENT
DATA SORT
DATA STORAGE/RETRIEVAL
ON-LINE DATA ACCESS

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Personnel Information at the David W. Taylor Naval Ship Research and Development Center is maintained on a computer disk storage system. The QUERY System, a software system used in maintaining the data base, and in deriving desired personnel information from that data base, is described. QUERY produces reports and tables based on the contents of the data base.

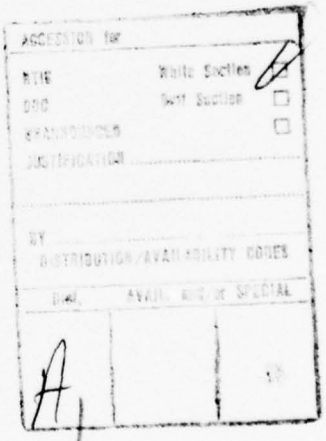
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES



INTRODUCTION

BACKGROUND

The David W. Taylor Naval Ship Research and Development Center (DTNSRDC) employs approximately 3000 persons, stationed both at Carderock and Annapolis in Maryland and at various detachments around the continental United States. To create and maintain personnel records and training records on such a large number of employees, a computerized record maintenance system, the Centralized Automated Personnel Management Information System (CAPMIS)[1], has been developed. CAPMIS generates a large number of personnel reports and automatically completes and prints personnel action forms (SF50's) each with correct information for the employee specified. CAPMIS is currently being used at the Naval Research Laboratory, the Capital Area Personnel Services Office - Navy, and the Navy Regional Finance Center, as well as at DTNSRDC.

To access information not presented in the standard reports produced by CAPMIS, the QUERY system was developed. This on-line interactive system is based upon a collection of programs and subroutines developed at the Naval Surface Weapons Center in White Oak, Maryland, and is included as a subsystem of CAPMIS. It provides a quick and efficient way of supplying management with up-to-date personnel information taken from the CAPMIS data base.

Documentation about the QUERY system was not provided by its developers. Since successful use of the system at DTNSRDC over a prolonged period requires the existence of some permanent source of information about the capabilities and use of the system, the present documentation effort was begun. In the process, several useful refinements to the QUERY system were suggested and implemented. A description of the improved system is provided in the pages that follow.

OVERVIEW OF THE DTNSRDC QUERY SYSTEM

The QUERY system is composed of a number of FORTRAN-coded programs and subroutines run on-line from a teletype terminal (TTY) interfaced with the Center's CDC 6700 SCOPE 3.4 computing facility. Two separate input files are used—one, a data file known as EMPDATA, structured from the COBOL-produced employee data file, which contains a separate record of data for each person employed at the Center; and the other, a file made up of various categories of employee data selected from the employee data file. The latter file, known as the Inverted File, is prepared by a program known as the Inverter. The actual programs and subroutines needed to interact with the input files and to carry out the retrieval and reporting functions indicated are furnished by the QUERY System. Under this system, output may be obtained either on-line at the TTY or off-line at the batch printer, although a TTY (either a high-speed 30 cps or a low-speed 10 cps) is used for input in any case. A block diagram of the QUERY System is provided in Figure 1.

---

[1] "Centralized Automated Personnel Management Information System, Personnel Operations Manual," (Prepared for the David W. Taylor Naval Ship Research and Development Center) Control Data Corporation (Dec. 1975).

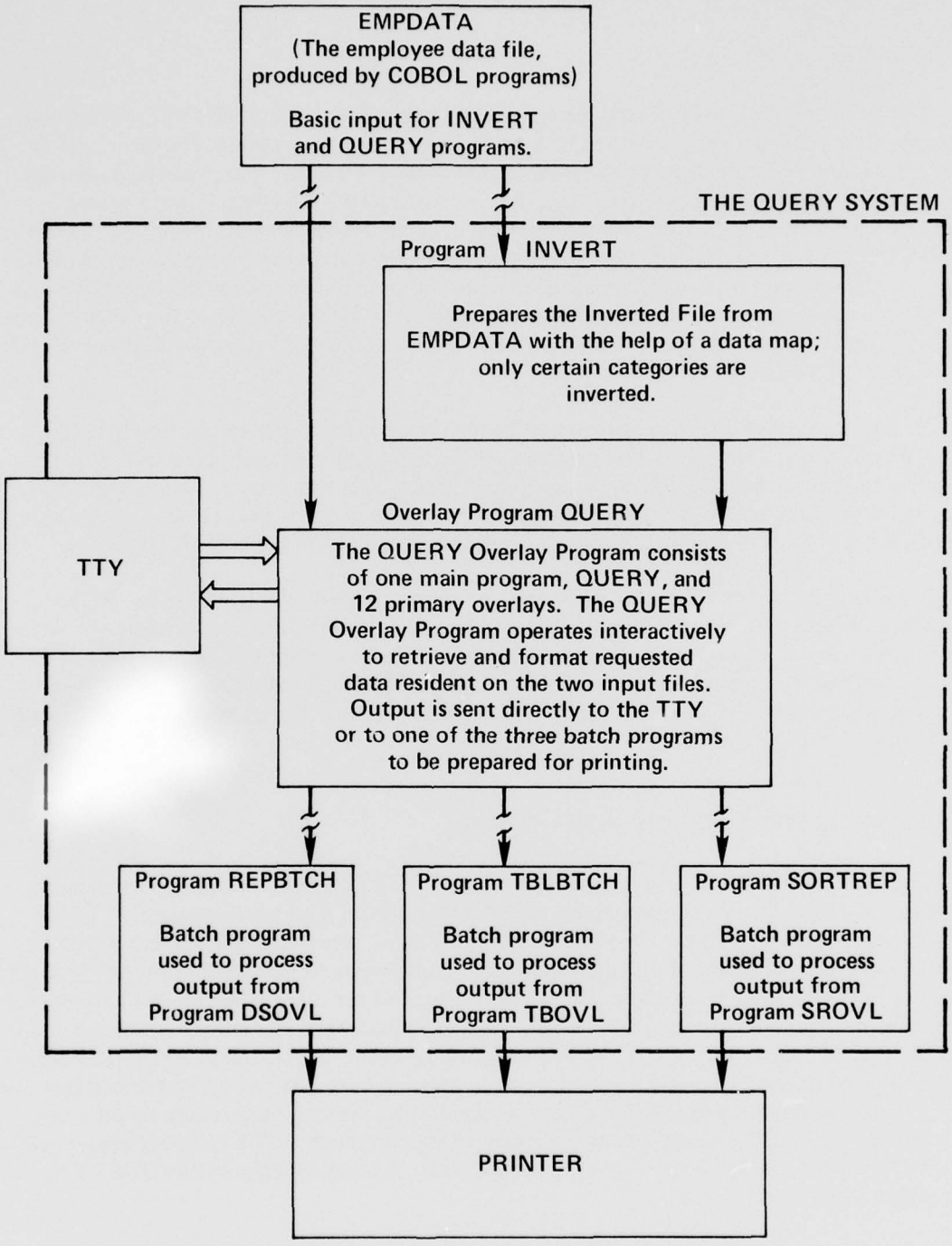


Figure 1 – Block Diagram of the QUERY System

The programs of the QUERY System are of three main types: (1) Those which prepare the data file, (2) those which select and format the data, and (3) those which produce batch output. The following paragraphs describe these programs in greater detail.

Preparation of the Inverted File

One main program and 13 subroutines are used for organizing certain employee information into a form suitable as input to the programs of the QUERY system. The main program, INVERT, builds lists of employee social security numbers, the arrangement of each list being based upon the values existing in the specific field in question (the Age field, for example) in the employee data records. Each list is based on a different field in the employee data records. All of these lists (records) are placed together on a single file to make up the Inverted File. At the same time that the lists are created, a directory to each of the lists is formed which serves as an index to the social security numbers of those employees who possess a specific value within the category (are of the same age, for example).

Inverted lists have been made up for a number of different categories, such as Age, Grade, Department, etc., which are frequently used to provide information. By creating these sorted records in advance, access time can be greatly reduced, since it is often just that one piece of information that is desired about a set of employees, not all of the information contained in the employee records. Details about the process of inversion are provided in the section of this report entitled "Preparation of the Inverted Data File."

Selection and Arrangement of Data

The QUERY Overlay Program is executed from a teletype terminal with all input typed into the computer by the user at the keyboard. The following commands will cause the program to select employees who meet certain specified qualifications and to display the names of those employees and related data about those employees:

GROUP and COMBINE	Determines all those employees having the qualifications specified.
DISPLAY	Displays the SSN's of those employees contained in each category.
ELEMENT	Displays requested data about a single employee.
SORT	Sorts employees within categories, using a specified sort key.
COMPUTE	Computes statistical data for specified sets of employees.
TABLE	Displays data for two categories in tabular form.

The user will ordinarily prefer to have his output typed out on-line at the TTY, although he may designate that it be printed off-line by the central site printer or at a remote batch terminal. Note that output from the SORT program cannot be directed to the TTY.

Production of Batch Output

Three programs are used for generating batch output: REPORT, SORTREP, and TABLE. Each is invoked by a control card record generated in QUERY and submitted to the batch queue of the CDC 6700. When executed, these control cards cause the following files to be accessed: the employee data file, the inverted file, and the file formed during the execution of QUERY. The programs carry out the following functions:

REPORT	Generates data about specified persons.
SORTREP	Generates a sorted list of employees, arranged according to given sort keys.
TABLE	Generates a statistical table which compares two categories.

USE OF THE QUERY SYSTEM

SIGNING ON FROM THE TELETYPE TERMINAL

The user "dials-up" the computer via a special telephone number. He then types in LOGIN, and follows this with his assigned USERNAME and PASSWORD. The TTY then prints out the word COMMAND to which the user responds COMRADE, PERSONNEL to indicate that the COMRADE executive[2,3] is to be invoked and the PERSONNEL subsystem entered. COMRADE then types out a series of question marks (? ? ? ?) in response to which the user types in the name of the specific procedure he wishes to enter. When the word QUERY is entered, the following files are attached:

- The main overlay program QUERYMAIN.
- The primary overlay program QUERYOVERLAYS.
- The indexed sequential employee data file (LFN = EMPDATA).
- The inverted employee data file.

QUERY then types out the question, ARE YOU FAMILIAR WITH THIS SYSTEM? If the user responds NO, a small tutorial session ensues to acquaint him with the steps to be taken. If he responds YES, control of the program transfers immediately to the subroutine EXEC of the main overlay.

The EXEC subroutine prints out the question COMMAND?, and the user types in the name of one of the following nine commands: GROUP, COMBINE, SORT, STOP, ELEMENT, TABLE, TIME, ABORT, or HELP. Each of the nine performs a necessary service to extract or

---

[2]Rhodes, T. and W. Gorham, "COMRADE–The Computer-Aided Design Environment Project, An Introduction," David W. Taylor Naval Ship Research and Development Center Report 76-0001 (1976).

[3]Tinker, R. and I. Avrunin, "COMRADE Executive System Users Manual," David W. Taylor Naval Ship Research and Development Center Report 76-0002 (1976).

tabulate information from the large employee file or to control the program. Examples to illustrate the use of these commands are provided in the section entitled, "Examples of the Use of the QUERY System." Each is discussed separately in this report.

QUERY COMMAND DESCRIPTIONS

The GROUP/COMBINE Commands

The GROUP and the COMBINE commands cause data from two or more different sets (*categories* in the inverted file) to be combined. Before either of the programs is called, control is first transferred to the subroutine CATEGRY in the main overlay. After the user enters a category name and specific values, CATEGRY calls the program VLOVL (Overlay (1,0)) to check on the validity of the values and to arrange them in a specified order. The resulting list, or collection of data, is used in the operation. To obtain other sets of data, the input string of commands and values is again searched for the next category name, and VLOVL is again called; the checking and sequencing process is repeated. When two files of SSN data have been selected, Overlay (2,0) is called and the appropriate subroutine (either GROUP or COMBINE) is entered.

The GROUP subroutine performs a logical AND operation on the two sets of data. Thus, by definition, elements – or SSN's – which occur in both sets are included in the new set. This process may be repeated for any number of input sets. Resultant sets are, in effect, themselves categories formed necessarily from certain of the categories in the inverted file. The result of each GROUP operation will always be a set which is no larger, and most likely smaller, than either of the input sets. After the last set of input data has been "grouped," a file of the desired data will exist in program storage.

The COMBINE subroutine performs a logical OR operation on two sets of data. Elements occurring in either set will appear in the result. The COMBINE operation may be repeated for any number of input categories. The set formed from certain of the categories of the inverted file itself becomes a new category which will be at least as large as the largest of the input sets, and perhaps larger. The file formed as a result of the operations of the GROUP and COMBINE subroutines can be operated upon by certain directives (DISPLAY and COMPUTE) which are issued at the terminal in response to the printout NEXT? – issued while CATEGRY is in control. These directives are described in the paragraphs that follow.

- The DISPLAY Directive. . . Displays data for a particular set of employees

This directive results in the calling of the program DSOVL (Overlay (3,0)) and the display of data concerning the employees contained on the file formed by a GROUP or a COMBINE operation. The printed output may be produced either on the teletypewriter or on the batch printer (consult the section entitled "Production of Batch Output"), the choice indicated while DSOVL is executing. Output may be produced in one of three predefined formats, called NAME, BRIEF, or DETAIL. The user may specify that a format of his own design be used. The special format will be named by the user and stored on disk until needed by DISPLAY to produce output during the current session.

In program INVERT, five words are associated with each field name stored in array FD. Of the approximately 250 fields available, as many as 100 may be used to define formats (FDDEF (5,100)). The five words associated with each field are as follows:

FDDEF (1,I) FDDEF (2,I)	Field name.
FDDEF (3,I)	CHAR | WORD; a pointer to the location of the field in the employee record.
FDDEF (4,I)	Size (in characters) of output line.
FDDEF (5,I)	M1 D1 M2 D2 M3 D3

The three predefined formats occur in memory as follows:

(1) BRF (2,10) [BRF (1,1) ∿ NAME (1,1)]

I	BRF (1,I)	BRF (2,I)
1	DIVISION-C	ODE
2	SERIAL	Δ
3	NAME	Δ
4	PAY-PLAN	Δ
5	SER	Δ
6	GRADE	Δ
7	STEP	Δ
8	BIRTH-DATE	Δ
9	SERVICE-CO	MP-DATE
10	SALARY	Δ

(2) NME (2,10)

I	NME (1,I)	NME (2,I)
1	SERIAL	Δ
2	NAME	Δ

(3) DTL (2,10)

I	DTL (1,I)	DTL (2,I)
1	SERIAL	Δ
2	NAME	Δ
3	SS	Δ
4	SERIES-TIT	LE
5	PD-NUMBER	Δ
6	HOME-STATE	Δ
7	HOME-PHONE	Δ
8	BLDG-NO	Δ
9	ROOM-NO	Δ
10	OFFICE-PH1	Δ

For use in conjunction with these formats, the following information is set up in program QUERY and stored in COMMON:

I	DISDEF (1,I)	DISDEF (2,I)	DISDEF (3,I)
1	BRIEF	10	a_1
2	NAME	2	a_2
3	DETAIL	10	a_3
	•		
	•		
	•		
24	•		

where DISDEF(1,I) is the name of the predefined format,
DISDEF(2,I) is the number of elements in the corresponding record, and
DISDEF(3,I) is the record number for format I.

As many as 21 new formats may be defined by the user. Information similar to that of the three predefined formats is stored in the array DISDEF (I,J) for each new format.

- The COMPUTE Directive . . . Determines average values of a set of elements

The issuance of this directive results in a call to the program CMOVL (Overlay (5,0)). This program operates upon the data of a base set (S) or category *which must previously have been set up in memory* (and stored on disk) by a COMBINE or GROUP operation. In response to the printout NEXT? – issued in CATEGRY, the user enters the word COMPUTE, followed by a category name and some value in that category. The category is either permanently contained in the inverted file or especially created during the current run. The first number to be computed and printed out is the number of elements, n, in the set S which have the value designated. Next to be computed and printed out is the sum of the values over all these elements, $\sum_{i=1}^{n} V_i$. Most likely some of the elements will have the same value. Finally to be computed and printed out is the average value over the elements, $(\sum_{i=1}^{n} V_i)/n$. An example of the use of the COMPUTE directive is included on page 37.

- The ELEMENT Directive . . . Displays data for a single employee

This directive, the coding for which occurs in the DSOVL overlay, is similar in function to the DISPLAY directive, except that data is displayed for one employee only. Further details about this directive/command are contained on page 10.

- The TIME Directive . . . Calculates the elapsed computer time

This directive results in a printout indicating the elapsed CP time and the CONNECT time for the current run. The directive (or command) may be entered whenever input is requested, and control is returned to the statement in the calling program immediately after the call to TIME.

- The HELP Directive . . . Prints out information on the use of the commands

The entry of the directive HELP results in a transfer of control to the subroutine CATHELP (category help) which provides assistance for the user of the program QUERY. CATHELP defines possible inputs and their corresponding outputs. In general, the HELP command may be entered whenever input is requested by QUERY. A specific HELP subroutine is available to give assistance to the user with respect to the particular subroutine in control. Return from a call to a HELP subroutine is made to the line immediately following the call. Further details about the HELP subroutines are to be found in the section entitled "On-Line Assistance on Use of Commands and Directives."

- The STOP and) Directives . . . Discontinue processing

Each of these directives produces varying results, depending upon what has been done previously and the responses to certain queries. If the category is to be saved, its "save name" is entered and then checked by the program. Names are arranged in alphabetical order, the key block is written out, and control words are set up. If the program is to continue, a new key block is read in and the new category is combined with another already in memory. If an exit from the subroutine CATEGRY and a return to the subroutine EXEC is desired, header and data blocks associated with key block KBA are released.

- The = Directive . . . Saves a data set

This entry causes the category to be saved on mass storage. A new key block is read in and, if desired, a new category is combined or grouped with an existing one.

- The ABORT Directive . . . Terminates the processing of the current command

This directive causes the current operation to be terminated and another command to be executed in the subroutine EXEC.

The SORT Command

The SORT command references the SROVL overlay which sorts a given category according to the values of its elements. The SORT command operates upon *cataloged categories,* which may be user-defined. When SROVL is entered, the following message is printed out at the TTY:

CATEGORY TO BE SORTED?–

The name supplied by the user is then checked against an array of acceptable names and against the special words HELP, TIME, and ABORT. If the directive HELP is entered, help on the use of the sort function is provided; if the directive TIME is entered, a printout of the elapsed time is produced; and if the directive ABORT is entered, the SORT program aborts. If the name supplied is none of these, and if it cannot be found in the array of acceptable names, the message

(name) DOESN'T EXIST

is printed out, another name is requested, and the entire procedure is repeated. If the category name is found in the array of names, the subroutine DISTART is called and a display format is selected. The user may develop his own format or select one of the standard types. If the word DEFINE is entered when the message ENTER DISPLAY TYPE is printed, the subroutine DEFINE is called. The user may enter names for fields to be included in his format and a name for the format itself. This information is stored for him as another display format. The user must also specify whether or not he wishes to have column headings printed.

The printout ENTER SORT TYPE is produced following the return from the display initiation subroutine DISTART. The name that indicates the predefined sort type (a key, or item of data) to be used in sorting the information is entered and the corresponding sort definition is read from disk. If the word DEFINE is entered, the subroutine SORTDEF is called which allows the user to make up his own sort definition from selected data fields. If the sort format specifies more than one field, the category data will be sorted on each field in turn, thus providing, in effect, a refinement of the sort in cases of "ties" or duplicate values in one field. The user can also specify in SORTDEF, for each field of the sort format, whether the values in a field(s) are to be sorted in ascending or descending order.

Only one predefined sort format is available. Named ALPHA, this predefined sort format consists of a single sort field, NAME. Using this format, the sort will be made on employee name. Sort formats are stored in the array SRTDEF(3,12), with three entries per definition: (1) name, (2) number of fields in the corresponding record, and (3) the number of the record containing the data. As many as 11 new sort formats may be defined by the user.

Since output from the program SROVL is always produced by a batch job, input for this job must be provided from disk records. The display definition and the sort definition are written to a disk record and the selected SSN's are written, one per record. The collection of records, or the disk file, forms the input to the batch job; the name of this file is generated by the subroutine THERE. The program SRTBTCH (Overlay (13,0)) submits the job to the batch queue. When the program SORTREP begins execution, it generates and prints out a sorted file.

The STOP Command

Entry of this command brings the QUERY session at the terminal to an end. The elapsed CP time and the CONNECT time are printed out, and control is returned to the COMRADE system (by which the entire QUERY system is controlled). The user may then select another subsystem of COMRADE, most usually LOGOUT, which will cause the termination of the user's session at the terminal.

The ELEMENT Command

This command is analagous to the ELEMENT directive entered in subroutine CATEGRY. Program control is transferred to DSOVL (Overlay (3,0)) and then to the subroutine ELEMENT. The purpose of this subroutine is to display information about one employee (SSN) at a time. First, the display is initialized by the subroutine DISTART. To the request for the destination of the output, the user responds with either HERE (for TTY output), or THERE (for batch output). An option for printing headings above the output is offered, and a choice of a format to govern the presentation of the data is given. As indicated earlier, three predefined formats are available (NAME, BRIEF, and DETAIL); the user may make up a format of his own, by typing in the word DEFINE. Then, in response to the program typeout ENTER DATA FIELD (S) or 'STOP' –, the user indicates the fields he wishes included in his output, and then follows this by the directive STOP. In response to the question NAME? –, the name of the new format is entered and the format is cataloged for the current run.

When control returns to the subroutine ELEMENT, the *element name* of an employee (actually his social security number) is requested. If the output is to be produced on the batch printer, the SSN is written to a disk file. If output is to be produced on the TTY, the employee record is retrieved by an IS (indexed sequential) call, using the SSN as a key. The subroutine PRINTER is called to extract, format, and print the pertinent data from the employee record.

Control is next transferred to a module of the code which will determine the next element name, and the same procedure is followed. Certain words entered in response to a request for input will trigger some definite action to be taken. In the case of batch output, the directive STOP will cause the job to be submitted to the central site; but in the case of TTY output, that directive will cause a direct return to be made from the program ELEMENT. The word ABORT causes the subroutine to be aborted. The word TIME causes the current time to be printed. The word HELP causes control to be transferred to a subroutine that will provide instructions for the use of the ELEMENT command.

The TABLE Command

Entry of the command TABLE causes TBOVL (Overlay (6,0)) to be called in. The program produces as output a table which specifies the number of employees having both the values of

category 1 and the values of category 2. Thus an NxM matrix is produced, n being the number of values in the first category, and m the number in the second.

Called from TBOVL, the subroutine TABLE first determines the destination of the output. The special words HELP, ABORT, and TIME may be entered at any time with the usual result. To the typed request ENTER CATEGORY NAME FOR ONE COMPONENT OF THE TABLE–, the user types in a category name. If the name is valid, a request for the second category is made. If the name cannot be found in the list of inverted categories, the statement

Name IS NOT A VALID CATEGORY NAME, TRY AGAIN

is produced, and the user enters another name. When both category names have been accepted, and if the output is to be produced HERE (at the TTY terminal), the table is generated by the subroutine TBLCNT. In the case of batch output, data for each category are written to a disk file (Tape 3) and submitted to the central site by program TBLBTCH, using a password generated by the subroutine THERE. The batch program TABLE makes use of the data file but does not use the employee data file EMPDATA. Further information relative to the batch program is provided in the section "Production of Batch Output."

The TIME Command

This command is issued whenever the user wants to know how much central processor time has elapsed, or exactly the amount of time used in the current terminal session since sign-on. After the coding has executed control returns to the statement following the one which requested the TTY input (TIME).

The ABORT Command

This command, which may be entered in many places throughout QUERY, returns control to the subroutine EXEC which prints out the message SESSION TERMINATED and indicates the amount of time that has elapsed. The EXEC subroutine also causes the disk files to be rewound and the indexed sequential system to terminate. The program stops and control is returned to the COMRADE executive system.

The HELP Command

This command may be issued at various times throughout the execution of the QUERY Overlay Program. As soon as it is issued, control is transferred to the subroutine which provides general information about the use of QUERY, or specific information about a certain function of the system. Control returns to the statement following that which requested TTY input.

PROCEDURE FOR SAVING CATEGORIES

Several unique sets of data, called "categories," are operated on by the QUERY system. These categories are formed from data contained in the employee data file using the process of "inversion" (explained later on in this report). Once formed, they are stored on a FORTRAN mass storage file. If the user wishes, he can construct his own categories, which will be cataloged for the duration of his session at the terminal only. Categories are used by the programs GCOVL,

DSOVL, SROVL, CMOVL, and TBOVL. The DISPLAY and COMPUTE subroutines, of the programs DSOVL and CMOVL, respectively, will operate only upon categories developed by the user within program GCOVL; SORT (SROVL) and TABLE (TBOVL) will operate on both types of categories.

The user forms a new category by selecting SSN's from given categories, using the GROUP or COMBINE operation. He may be as definitive as necessary. After a category has been constructed, the message SAVE CATEGORY – will be printed out by the program. If the user responds YES, the message ENTER CATEGORY SAVE NAME – is printed out, and the user must type in an identifying name for his new category. When the category is added to the mass storage file, its name is added to the array of category names contained in core memory, in the correct alphabetic position. Memory space for 64 category names and associated block names is set aside in the QUERY program. The corresponding key block is written to mass storage along with header block indicators and data blocks.

ON-LINE INFORMATION ABOUT QUERY COMMANDS AND DIRECTIVES

The QUERY user may request help, in the form of explanations and guidance, at various points in the executing program. The entering of the command HELP will result in printed instructions pertinent to the particular process currently being undertaken. The main overlay calls upon several HELP routines. The QUERY system is so arranged that all the HELP routines are concentrated in two separate primary overlays. Program HLPOVL1 (Overlay (7,0)) is called whenever help with a particular category is needed (CATHELP) or when help in specifying values is needed (SPCHELP and VALHELP). Since HLPOVL1 is a primary overlay, it has access to all of the subroutines of the main overlay. Other subroutines needed – CMBHELP, CMPHELP, DISHELP, ELEHELP, SETLOOP, and NUMBER – are included in the HLPOVL1 program directly. Program HLPOVL2 (Overlay (10,0)) is called whenever general information about the whole system is desired (COMHELP). The program HLPOVL2 contains all of the subroutines COMHELP needs–namely, CMBHELP, ELEHELP, GRPHELP, SRTHELP, and TBLHELP. Like HLPOVL1, Program HLPOVL2 also may reference all of the subroutines of the main overlay.

Each of the overlays ((1,0) - (6,0)) calls on HELP subroutines. Those needed by each program are included in the respective overlay. All told, there are 12 HELP subroutines (13 entry points). A description of each follows:

CATHELP – Describes the Subroutine CATEGRY

As with all HELP subroutines, and, indeed, with the entire QUERY system, CATHELP operates in a tutorial fashion. When a question is typed out at the TTY by the system, the user responds by typing in his answer. The user may request the definition of any category named on the category list. A list of the directives to subroutine CATEGRY may also be requested as output. The user may request information from CATHELP about any of the several operations CATEGRY initiates, including GROUP, COMBINE, DISPLAY, COMPUTE, and ELEMENT. A HELP subroutine furnishes explanations of each of the aforementioned operations.

CMBHELP – Describes the COMBINE Command

This subroutine will explain the use of the COMBINE function of the (2,0) overlay, GCOVL. The steps needed to combine portions of categories are described. CMBHELP explains the formation of files to be used immediately (e.g., by DSOVL or by CMOVL), and the files to be cataloged as new categories and used at a later time.

CMPHELP – Describes the COMPUTE Directive

This subroutine provides instruction on the use of Overlay (5,0), CMOVL. CMPHELP can be called either from the subroutine CATEGRY in the main overlay or from the subroutine COMPUTE in the (5,0) overlay to explain the method of computing the number of elements having a given attribute and the method for computing the average value of the attribute over all the elements possessing it. The input for a COMPUTE directive consists of a category name. When the CMOVL program is executing the HELP, ABORT, and TIME directives may be issued. These inputs were described in the section on GROUP/COMBINE commands.

COMHELP – Provides an Overview of the QUERY System

This subroutine, called either from the main routine QUERY or from the executive subroutine EXEC, provides general information about the entire QUERY system. It offers general textual information on the use of the GROUP and COMBINE directives and briefly explains the STOP, TIME, ABORT, and HELP commands. Requests for additional information about other commands will result in a transfer to one of the following appropriate subroutines:

Command Name Entered	Subroutine Called
GROUP	GRPHELP
COMBINE	CMBHELP
SORT	SRTHELP
TABLE	TBLHELP
ELEMENT	ELEHELP

Each of these subroutines explains the use of the command indicated. The user may issue any of the above-named commands while COMHELP is executing.

DEFHELP – Describes the Definition of Formats

DEFHELP is used as an entry point of subroutine DISHELP, and is called from the subroutine DEFINE in the overlay DSOVL by the user when help in defining his own display format is needed. Subroutine SETLOOP is called to compute parameters to control the printing loop which prints out the names of fields and the widths (in characters) reserved for these fields. With this information, the user is able to make up his own individual format(s).

DISHELP – Describes the Display of Data

This subroutine may be called either from the subroutine CATHELP in the main overlay or from the subroutine DISTART in the third primary overlay, DSOVL. The DISHELP subroutine explains how the DISPLAY directive is used to select a display format. The user may request a printed list and a printed explanation of defined formats which will list the fields in the format.

ELEHELP – Describes the ELEMENT Command

This subroutine explains the use of the ELEMENT command. The subroutines used to display information concerning individual elements (persons) composing the data base are located in the (3,0) overlay program DSOVL. The individual's social security number is the key which is used to retrieve information about that person from the data base for subsequent display.

GRPHELP – Describes the GROUP Command

This subroutine explains the GROUP portion of Overlay (2,0), GCOVL. The GROUP subroutine compares two or more categories and selects elements common to all categories. Subroutine GRPHELP describes the input to the GROUP command and explains how the results of the operation may be saved as a new category.

SDFHELP – Describes the Definition of a SORT Format

This subroutine explains how SORT formats are defined. These formats, used by the (4,0) overlay SROVL, specify which fields are to be sorted. Categories of data are sorted, according to each format in turn, in either ascending or descending order. "Ties," or elements having a common value in some category, are resolved in this way.

SPCHELP – Describes the Specification of Data

This subroutine explains how values of a category are specified. It is called from the subroutine CATEGRY in the main overlay when the user types in the word HELP after having entered a category name. SPCHELP specifies the ways in which values can be combined to produce a new set.

SRTHELP – Describes the SORT Command

This subroutine explains how categories are sorted. It is called from the SORT subroutine in the overlay (3,0), SROVL.

TBLHELP – Describes the TABLE Command

This subroutine, called from Overlay (6,0), TBOVL, explains how to generate a table that will compare the values of one category with those of another.

VALHELP – Explains the Choice of Input Values

This subroutine, called from the main overlay, explains how values (all or only part) of a category are specified, and how the number of elements having a particular value are determined.

PREPARATION OF THE INVERTED DATA FILE

The Employee Data Record

The *employee data record* is formated as follows:

	CHARACTER									
WORD	0	1	2	3	4	5	6	7	8	9
0	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	b_0
1	b_1	b_2	b_3	b_4	c_0	c_1	c_2	c_3	d_0	d_1
2	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}
3	d_{12}	d_{13}	d_{14}	d_{15}	d_{16}	d_{17}	d_{18}	d_{19}	d_{20}	d_{21}
4	d_{22}	d_{23}	d_{24}	d_{25}	d_{26}	d_{27}	e_0	f_0	f_1	f_2
.										
.										
.										
63				.	.	.	z_1	z_2	z_3	z_4

A record of this type exists in the data base for each employee. Each record is identified by the first item of data in the record (positions a_0 - a_8) which is the employee's social security number, used by the index sequential system in accessing the record. When attached to the QUERY program, the file of employee records is known as the EMPDATA file. Since the records are accessed by their SSN's, their position's on the file are irrelevant. The data field (d_0 - d_{27}) contains the octal representation of the name of the employee. Obviously, the fields need not necessarily end at word boundaries, and may be longer or shorter than one word in length.

The employee records are generated by a COBOL program. The COBOL program also generates a *data map* that points to each field of the employee record. Only one data map is needed, since the order of the data fields is the same in each data record. The program INVERT forms individual 5-word arrays, one for each field in the record.

Function of the Program INVERT

The program INVERT – which consists of a main routine, ten FORTRAN subroutines, and three COMPASS subroutines – operates upon an input file of employee data records (EMPDATA) to produce a so-called "inverted" file. Although the term "inverted" is not applied in a traditional sense here, it's use to indicate a special kind of file (made up of categories of information) is widely recognized.

Consider the employee data records, each of which contains 64 different items of information (fields), some used as search criteria more frequently than others. (Corresponding fields contain corresponding information in each of the records). Time required to access records of employees having given values of certain categories (fields), can be significantly reduced by rearranging the employees according to selected categories, i.e., category AGE used for one inverted list, the category GRADE for another, etc. – the file of records is "inverted on these categories."

For each chosen category, the records (as identified by the SSN's) are arranged in some logical order of values of field. Figure 2 illustrates the results of inversion for one category. In addition to forming the data records, INVERT produces a "header block" for each inverted category. The header block serves as a directory to the data. A two-word entry is made for each value of the category. The first word specifies the value. The second contains two entries, one pointing to the first element of the data blocks having the header value and the other indicating the number of data elements having the value. Diagramatically, the two words have the following form:

Word 1	Value	
Word 2	Pointer to first element having specified value.	Number of elements having this value.

In the example given in Figures 2 and 3, the employees are classified by age; the SSN's of all employees of the same age are grouped together. Within each age group, the SSN's may be further organized in ascending order. The resulting "inverted" file is written out to a disk file.

Operation of Program INVERT

Subroutine DATADIV of the program INVERT generates a five-word array for each of the fields ($\leq$250) contained in the COBOL input data record. The number of fields (divisions) contained is returned from DATADIV in location NUMDIV. The arrays take the following form:

Array Word	Contents of Location
FMT (1,I) FMT (2,I)	Name of field
FMT (3,I)	CHAR | WORD A pointer to the corresponding field in the employee record.
FMT (4,I)	Size of the output line (in characters)
FMT (5,I)	Data needed by the subroutine FETCH for retrieving a field from the user record.

After subroutine DATADIV has finished executing, control is returned to INVERT. The 5-word subarrays formed by DATADIV are alphabetically arranged by name of field within the array FMT (5,256) which is written to a mass storage file, and the indexed sequential (IS) data storage and retrieval system is then initialized. The card input, consisting of two cards per category, is then read; the first contains values for the nine variables that define the field, and the second contains an explanation of what the field represents. The existence of the category name in the FMT array is verified. Details of the card input follow:

Variable	Format	Description of Variable
Card 1:		
COBNAME	A20	Name of the entire field in the *COBOL input* (the employee data file)

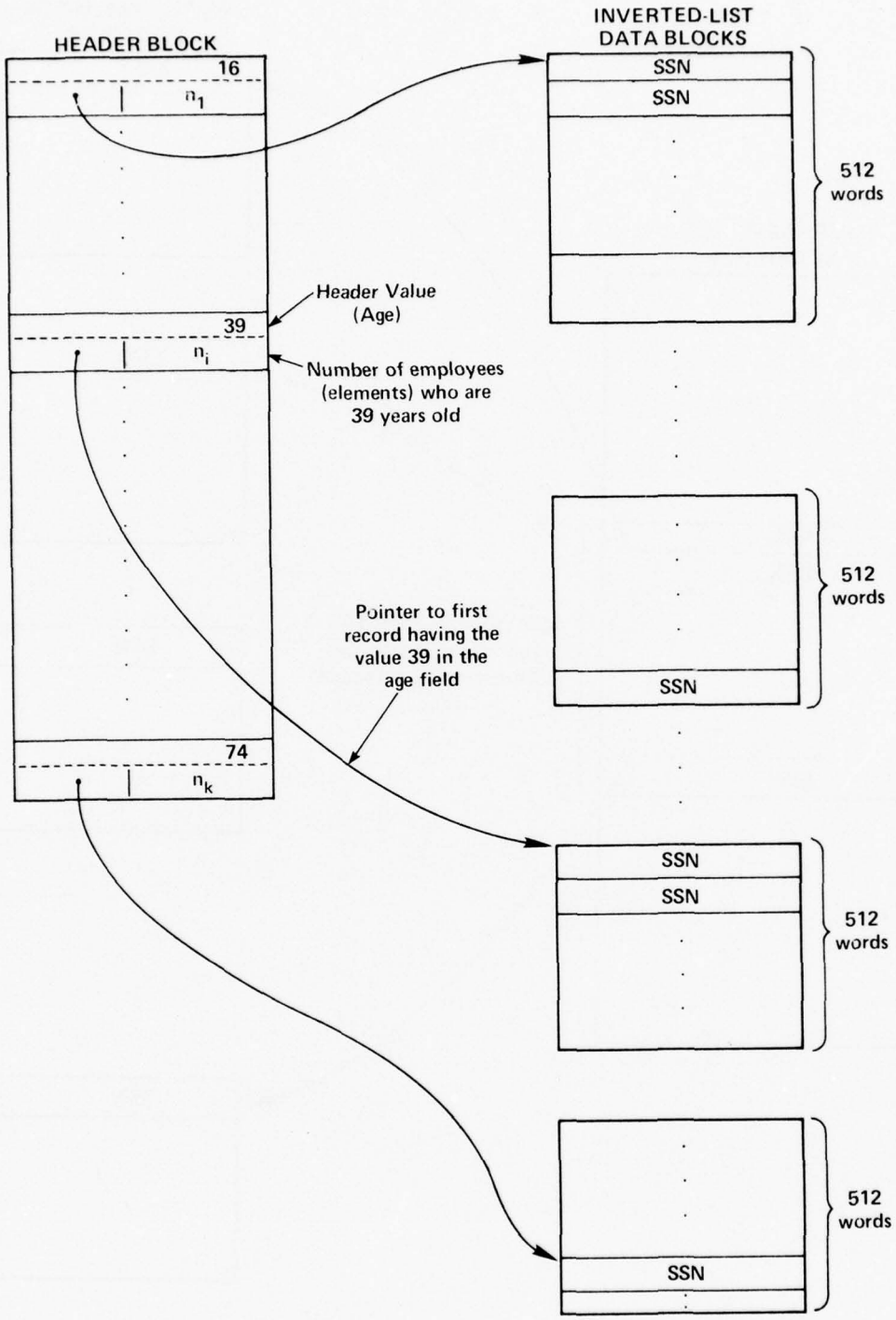


Figure 2 – Structure of Inverted List for the Category AGE

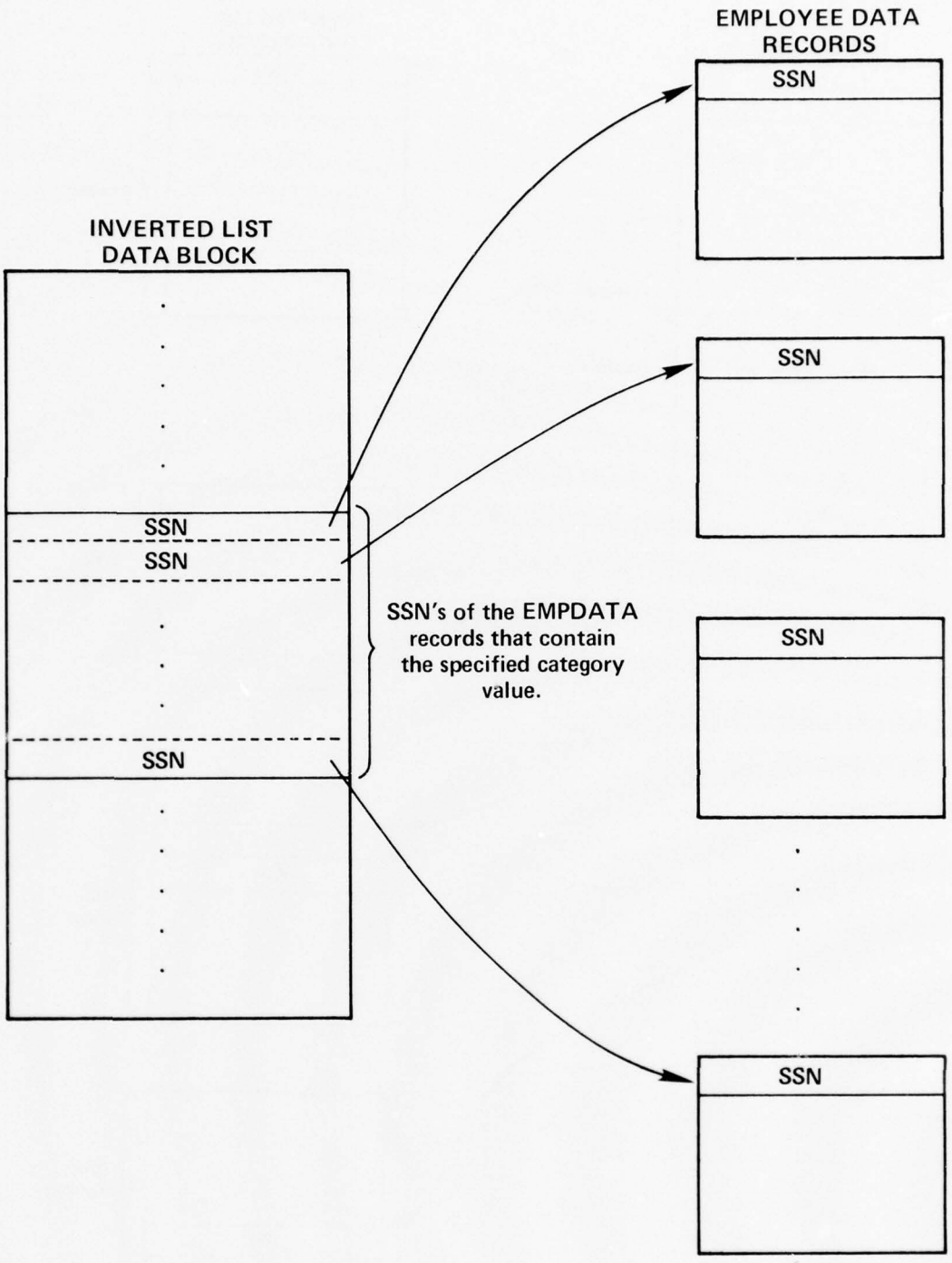


Figure 3 – The Relationship between an Inverted-List Data Block and the Corresponding Employee Records

Variable	Format	Description of Variable
Card 1–Continued		
QUERYNM	A20	Name of the subfield (extracted from COBNAME) in the QUERY program which is used in the inverted file.
FLDLEN	I10	Total length of the input field in the EMPDATA user record
SCHAR	I5	The relative position of the subfield within COBNAME
LENS	I5	Length of subfield
SPACE	I1	Indicator (used in GETDATA): Set to "1" if field is blank (Δ ———— Δ)
ZERO	I1	This variable is examined within the subroutine VALUE of Overlay (1,0); if value is other than zero, the subroutine FORGIVE is called to fill the field QUERYNM with zero's.
TIME	L1	An indicator: If T, date (Julian) is to be converted to a number of years (up to the present); otherwise, indicator is set to F.
VARSCAN	L1	If a blank occurs within the first word of the field retrieved from EMPDATA, and if VARSCAN = T, the remaining characters in the field are filled with blanks. If VARSCAN = F, the field remains as it is.
NUMERIC	I1	Blank
Card 2:		
EXPLN	7A10	Seven (or fewer) words explaining the category represented by the field

The following table shows four sample inputs:

	Example 1	Example 2	Example 3	Example 4
COBNAME:	DIVISION-CODE	DIVISION CODE	BIRTH-DATE	SEX
QUERYNM:	DEPARTMENT	ORGCODE	AGE	SEX
FLDLN	6	6	0	0
SCHAR	1	1	Δ	Δ
LENS	2	4	Δ	Δ
SPACE	0	0	0	0
ZERO	1	1	1	0
TIME	F	F	T	F
VARSCAN	F	F	F	F
NUMERIC	Δ	Δ	Δ	Δ

Explanations of the examples follow:

Example 1. The total field-length for DIVISION-CODE is FLDLEN=6. The length of the subfield DEPARTMENT is LENS=2; the subfield starts in the first character of the field SCHAR=1. Since ZERO=1, the entire (sub) field is checked for blanks, from the most significant end. If one is found, the remaining characters in the word (to the total of KB(5)) are filled with zero's. Since TIME=F, indicating that no date has been given, no conversion is required.

Example 2. The length of subfield ORGCODE is LENS=4; the subfield starts in the first character of the field DIVISION-CODE. The (sub) field is checked for blanks (ZERO=1). If one is found, the field is filled with zero's.

Example 3. Since FLDLN=0, SCHAR and LENS are not examined. Since TIME=T, the increment of years, computed in another subroutine, is stored in the first two digits of KB(5).

Example 4. Since ZERO=0, no check for blanks is made. It is assumed that the field for category SEX is filled.

Information from the first input card is used to form some of the elements of a key block as follows:

KB(1)	SPACE | ZERO
KB(3)	NUMERIC
KB(4)	MULT (1) · MULT (2) · MULT (3), where the MULT(I) are extracted from FMT (5,I)
KB(5)	Length of the field or subfield for the current format, normally FMT (4,I); if TIME=.T., KB(5)=2; if VARSCAN=.T., KB(5)=10; if FLDLN=0, KB(5)=LENS.

The second input card supplies information for the definition of the last seven words of the key block.

The FMT array is read from disk and then the COMPASS subroutine SORTER is called, which, in turn, calls the System SORT/MERGE subroutine, SMCON7. The sorting of all the employee records takes place in SMCON7, with the assistance of two user-coded subroutines, GETDATA and PUTDATA. The former is used to retrieve from the employee's record (1) the specific field indicated by the format (for example, age), and (2) the employee's social security number. These items of information make up a separate two-word array which is then placed in with the arrays for other records in ascending alphanumeric order by SMCON7. After all of the two-word arrays have been retrieved and stored in their correct sorted position, the subroutine PUTDATA is called to write the array to a disk file. Key control data for the current category is made up in memory, and the key block is written to disk.

The process just outlined is repeated for each category. After all of the categories have been sorted and all data have been written to disk, the disk information is edited and printed out, according to the format indicated. The IS system is then terminated, and the program INVERT ends.

Format and Definition of a Key Block

Key Block Word Numbers	Contents of Word
KB(1)	Values for SPACE and ZERO; taken from input card
KB(2)	Number of header values for the category
KB(3)	Value for NUMERIC; taken from INVERT
KB(4)	Value of MULT(1)·MULT(2)·MULT(3); set in INVERT
KB(5)	Number of digits in value
KB(6)	Number of records reserved for header blocks; each entry indicates the number of elements per value.
KB(7)	Total number of records of SSN's which make up this category
KB(8) ⋮ KB(15)	Record number of each record of SSN's contained in this category
KB(16) ⋮ KB(J)	Record numbers of the header-block records
I_1	First value of first header block
I_2 ⋮	Last value of first header block
I_n	Last value of last header block
D_1 ⋮ D_7	Seven words that define the category

Format of the Inverted File

The categories mentioned in the following format description are subject to reordering or removal by the persons in charge of the DTNSRDC personnel program. Therefore, except for the first three records, the record numbers shown may be different from those actually appearing on an inverted file, both at this installation and at others.

Number of Words	Record Number	Variable Name	Description of Variable
3	1	KEY(CNTRL(1))	Number of keys
		NUMDIV(CNTRL(2))	Number of defined formats
		DATE(CNTRL(3))	Date of inversion
3*KEY	2	KCB(1,KEY_1)	Query name
		KCB(2,KEY_1)	Number of words in key block
		KCB(3,KEY_1)	Key-block number
		⋮	
		KCB(1,KEY)	
		KCB(2,KEY)	
		KCB(3,KEY)	
5*NUMDIV	3	FMT(1,1)	
		FMT(2,1)	The format for Element 1 of all the defined entries in the employee file; e.g., the social security number
		FMT(3,1)	
		FMT(4,1)	
		FMT(5,1)	
		⋮	
		FMT(1,NUMDIV)	
		⋮	
		FMT(5,NUMDIV)	
8*512	4-10, 12		Records available for SSN's of category QUERY $NAME_1$ (DEPARTMENT)
512	11		Header block
64	13		Key block for DEPARTMENT
8*512	14-18, 20,21,23		Records available for SSN's of category QUERY $NAME_2$ (ORGCODE)
2*512	19,22		Header blocks
64	24		Key block for ORGCODE
8*512	25-31,33		Records available for SSN's of category QUERY $NAME_3$ (AGE)
512	32		Header block

Number of Words	Record Number	Variable Name	Description of Variable
64	34		Key block for AGE
8*512	35-41,43		Records available for SSN's of category QUERY $NAME_4$ (LENGTH)
512	42		Header block
64	44		Key block for LENGTH
8*512	45-51,53		Records available for SSN's of category QUERY $NAME_5$ (SEX)
512	52		Header block
64	54		Key block for SEX
8*512	55-61,63		Records available for SSN's of category QUERY $NAME_6$ (SCHEDULE)
512	62		Header block
64	64		Key block for SCHEDULE
8*512	65-71,73		Records available for SSN's of category QUERY $NAME_7$ (PAYPLANCOD)
512	72		Header block
64	74		Key block for PAYPLANCOD
8*512	75-81,83		Records available for SSN's of category QUERY $NAME_8$ (SERIES)
512	82		Header block
64	84		Key block for SERIES
8*512	85-91,93		Records available for SSN's of category QUERY $NAME_9$ (GRADE)
512	92		Header block
64	94		Key block for GRADE
8*512	95-101,103		Records available for SSN's of category QUERY $NAME_{10}$ (ROLLSTATUS)
512	102		Header block
64	104		Key block for ROLLSTATUS
8*512	105-111,113		Records available for SSN's of category QUERY $NAME_{11}$ (CEILINGCOD)
512	112		Header block
64	114		Key block for CEILINGCOD

Number of Words	Record Number	Variable Name	Description of Variable
8*512	115-121,123		Records available for SSN's of category QUERY NAME$_{12}$ (SUPERVISOR)
512	122		Header block
64	124		Key block for SUPERVISOR
8*512	125-131,133		Records available for SSN's of category QUERY NAME$_{13}$ (PROFCODE)
512	132		Header block
64	134		Key block for PROFCODE
8*512	135-141,143		Records available for SSN's of category QUERY NAME$_{14}$ (LOCATION)
512	142		Header block
64	144		Key block for LOCATION
8*512	145-151,153		Records available for SSN's of category QUERY NAME$_{15}$ (XYZ)
512	152		Header block
64	154		Key block for XYZ
8*512	155-161,163		Records available for SSN's of category QUERY NAME$_{16}$ (PROMO-TYPE)
512	162		Header block
64	164		Key block for PROMO-TYPE
8*512	165-171,173		Records available for SSN's of category QUERY NAME$_{17}$ (EDUCLEVEL)
512	172		Header block
64	174		Key block for EDUCLEVEL

Category Definitions

Not all data fields available on the employee file are inverted. Moreover, the number of categories defined may be reduced or increased subject to the limitations of the QUERY program(s). The following categories are currently in use at DTNSRDC.

Name of Category	Basis of Category
DEPARTMENT	Department where employed
ORGCODE	Organization where employed
AGE	Employee age, in years
LENGTH	Length of government service
SEX	Sex: 1, male; 2, female
SCHEDULE	Type of work schedule: 1, full-time 2, part-time 3, intermittent
PAYPLANCOD	Pay plan
SERIES	Work series or occupation
GRADE	Employee grade (two-digit)
ROLLSTATUS	Status of employee on rolls: 1, on; 2, off; 3, LWOP; 4, military
CEILINGCOD	Ceiling of employee position; FTC, TPT, etc.
SUPERVISOR	Supervisory status of employee: 1, supervisor; 2, non-supervisor
PROFCODE	Professional status
LOCATION	Physical location of employee
XYZ	(Category not yet defined)
PROMO-TYPE	Type of promotion received
EDUCLEVEL	Educational level attained

PRODUCTION OF BATCH OUTPUT

General Description

The QUERY system enables the user to obtain his output either as a batch-run printout or as a teletype printout. The production of batch output requires the use of an initialization program, subroutines to formulate names for data files generated by QUERY, and programs to submit jobs to the batch queue. At some time subsequent to the use of QUERY, the programs REPORT, SORTREP, and TABLE are used to produce the desired output from the generated input files.

The BTCH program (QUERY Overlay (11,0)) is always called from the main program (Overlay (0,0)) when QUERY begins executing. BTCH retrieves and reformats certain parameters which are needed to complete images of job control cards. Since the QUERY system is used by a number of different activities, each with its own CAPMIS data base, the parameters retrieved (from the COMRADE Executive) by BTCH are unique to a particular activity's data base. BTCH sets up array IARRAY which includes the job order number for the batch job, the permanent file name (PFN) of the employee data file, and passwords necessary to access this file. The array IARRAY may be used for all three types of batch output, and is stored in a COMMON block; the BTCH program need not be called again.

In the programs DSOVL (Overlay (3,0)) and TBOVL (Overlay (6,0)), the user at the TTY terminal is asked where he would like his output printed. If he answers HERE, output is directed to the teleprinter; if THERE, output is produced by a batch run. When batch output is requested by any one of the three programs a subroutine is called to generate a unique name for the input data file. The name, consisting in part of the current date and time, forms a PFN for the data input file used when the file is attached during a run of the batch program. (When created by QUERY, the local file name of this file is TAPE3.) The data input file is generated by a subroutine within DSOVL, SROVL, or TBOVL.

As soon as the data file is complete another primary overlay (REPBTCH, SRTBTCH, or TBLBTCH, respectively) is called in to rewind, catalog, and unload the file. The batch control card record is completed, written to the batch input file, and submitted to the batch input stream. The user must decide whether his printout will be produced at the central computer site or at some remote user terminal.

Options Available

- The REPORT Program

The REPORT program is located on a permanent file on CDC 6700 computer disk storage. It can generate a "report" consisting of tabulated information about a set of employees either on a central site printer or on a remote user terminal. The set of employees to be reported on and the format to be used are specified by the user at a teletype terminal during the execution of OVERLAY (3,0) of the QUERY system. Data generated by QUERY for the REPORT program are written to a disk file (LFN=TAPE3) and accessed by means of ordinary FORTRAN read statements. The employee file (LFN=EMPDATA) is referenced by indexed sequential (IS) calls. The REPORT program itself is attached under the local file name (LFN) RPT. This program does not access the inverted file.

The REPORT program reads the display format from the TAPE1 file which consists of a number of predefined fields (for editing purposes) and social security numbers (SSN's). The SSN's are read, one at a time, into the location TAG, and the corresponding 63-word employee records are read from EMPDATA. To produce output (on TAPE3), the subroutine PRINTER is called. PRINTER calls upon the subroutine FETCH to extract certain fields of data from each employee record, according to the display format designated, and then it prints out data, with or without headings, at the central site printer or a remote user terminal, whichever has been specified. (FETCH and PRINTER are subroutines of the REPORT program.)

The entire REPORT program is comparable to that part of OVERLAY (3,0) (program DSOVL of the QUERY system) which produces output on the teletype printer. The operation of REPORT is represented by the block diagram of Figure 4.

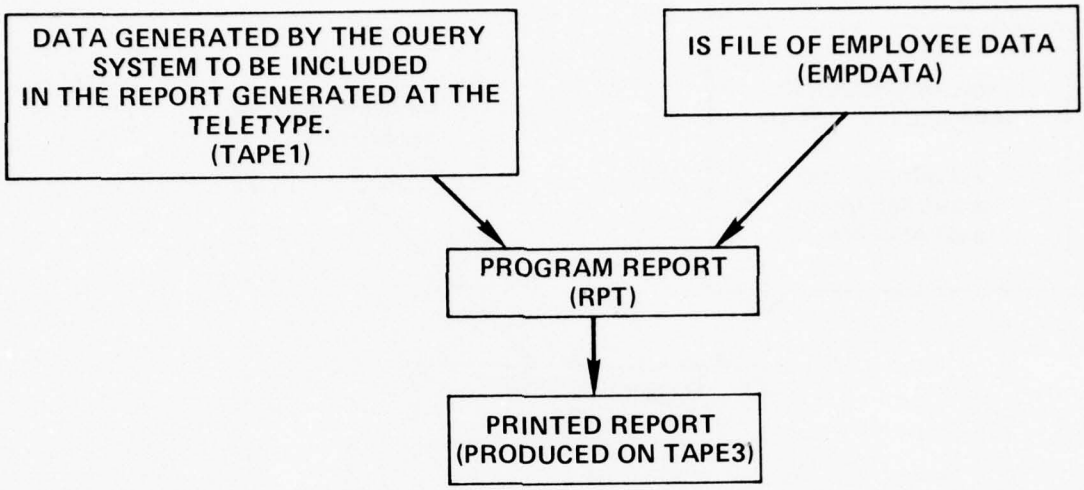


Figure 4 – Operation of Program REPORT

- The SORTREP Program

The SORTREP program (Figure 5) is used to produce a sorted list (or report) of a set of employees by means of a batch run, each employee designated in a data file by his SSN.

The SORTREP program is stored as a permanent file on 6700 computer disk storage, and is attached during a run under the local file name (LFN) SRT. Input to the program is generated by the user at the teletype terminal during the execution of OVERLAY (4,0). The input–consisting of the SSN's of all employees to be represented on the list, a display format to indicate the fields to be displayed, and a sort definition that specifies which fields will be sorted–is read from TAPE1 by the program SRT, using ordinary FORTRAN read statements. A display definition consists of five items (words) of data (page 29) which enable the program to retrieve specific data fields from the file. The data retrieved are used to format print records which constitute the output for selected employees. The sort definition is a slightly modified version of a display format in that the fifth word contains an alphanumeric A which indicates an ascending sort. Within SORTREP, the sort definition is slightly modified; the number of the first byte (or character) in the sort field is placed in the second word of the format. The resulting sort definition has the form shown on page 29.

The following diagram indicates the actions performed during a sort:

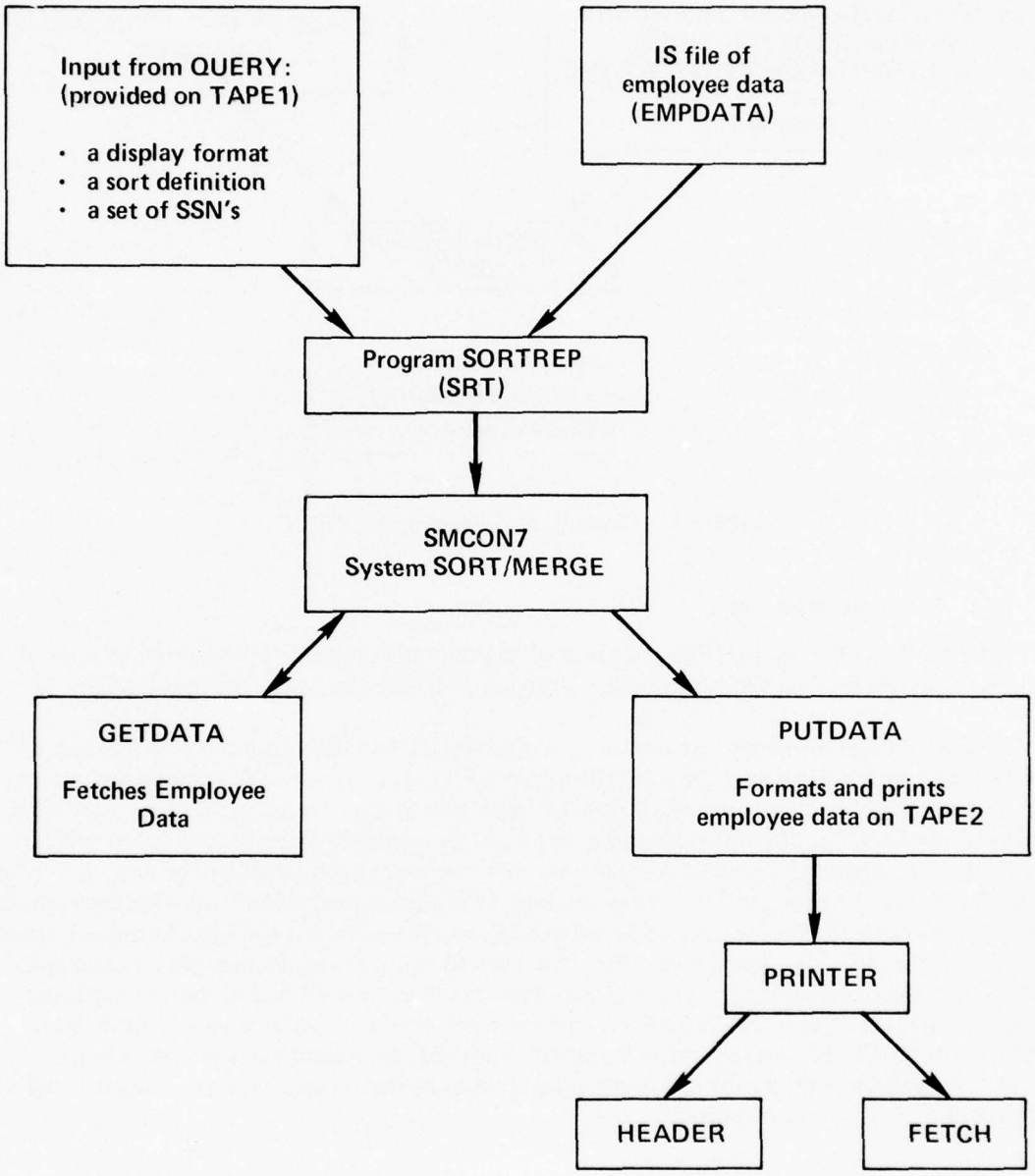


Figure 5 – Operation of Program SORTREP

Word	Contents	Example
1	Name of the field	NAME
2	Number of the first byte or character of the sort field	11
3	Starting character/starting word	0——0 | 0——2
4	Number of characters in format	28
5	Indicator of ascending or descending order of sort	A△————△

If more than one sort definition is specified in a sort job, each field is sorted successively. In this way, "ties" that occur when two or more employees have the same value for a particular field can be resolved.

The employee file, which contains a record of data for every employee, is attached to the batch job as EMPDATA. For each SSN contained in the input a corresponding data record is retrieved, using calls to the IS subroutines (Appendix D). From each 63-word data block, the desired information is extracted and passed to the subprogram SORTER. The 6700 SORT/MERGE system used by SORTREP does the actual work of sorting. The SORTREP program calls upon the GETDATA subroutine to access data, and the PUTDATA subroutine to write out sorted data. The subroutines PRINTER, HEADER, and FETCH control the output process. Program SORTREP does not access the inverted file.

- The TABLE Program

The TABLE program (Figure 6) is used to produce a statistical table that relates two inverted categories of QUERY. The input data file for the program is generated in OVERLAY (6,0) of QUERY, and is attached to and read by program TABLE as TAPE2. Program TABLE itself is attached to the job under the local file name TBL. Since this program deals with categories of employees rather than individual employees, the employee data file and indexed sequential subroutines are not used.

The program operates in the same way as some parts of the (6,0) overlay of the QUERY program. The table printed out indicates the numbers of elements (employees) having the header values of both the first and the second categories. Thus, a matrix of values is produced in which the columns represent the headers of one category and the rows representing the headers of the other.

The input data consists of (1) header blocks for each of two categories, (2) key blocks, and (3) the names of the two categories to be tabulated. The final results of the TABLE program are written on file TAPE6, or they are diverted to a user terminal.

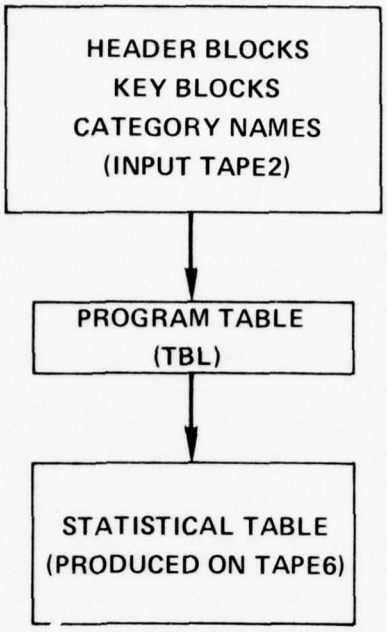


Figure 6 – Operation of Program TABLE

EXAMPLES OF USE OF THE QUERY SYSTEM

Following are several examples which illustrate the QUERY capabilities and the use of the batch programs REPORT, SORTREP, and TABLE. Since the programs and subroutines can be used in numerous combinations, these examples are by no means exhaustive.

Example 1 – Use of the GROUP command and the display directive:

The following example illustrates how sets of certain employees are selected, each additional category further restricting the set. The resultant set is displayed on the teletype printer in the NAME format (serial number and name of employee).

```
COMMAND?–GROUP, DEPARTMENT, 18, AGE, 25, GRADE, 11, SEX, 1
     XXX ELEMENTS WERE FOUND
      YY ELEMENTS WERE FOUND
      ZZ ELEMENTS WERE FOUND
      WW ELEMENTS WERE FOUND
NEXT?–DISPLAY, HERE, NAME
  xxxx     nnn . . . n
  xxxx     nnn . . . n
    .
    .
    .
```

Since the set contains only 25-year-olds to begin with, a query for 26-year-olds would result in the following exchange:

NEXT?=AGE, 26

0 ELEMENTS WERE FOUND

NO ELEMENTS FOUND

COMMAND?–STOP

By entering the word STOP instead of another command name, the QUERY program is told to print out the following information and then terminate:

SESSION TERMINATED

ELAPSED CP TIME = xx.xxxx

CONNECT TIME = x.xx.xx

STOP

Example 2 – Error messages and the response to a request for help:

COMMAND?–AGE, 21

AGE IS AN INVALID COMMAND

In this example, the command name was omitted, so the system interpreted the word AGE as a command. Since only the name GROUP or COMBINE would have been acceptable, the word AGE caused an error message to be printed. In the exchange

COMMAND?–GROUP, AGE2

AGE2 INVALID CATEGORY OR DIRECTIVE

the command name GROUP is acceptable but the AGE2 is not, since AGE2 is neither a category nor a directive. In the terminal exchange

NEXT?–AGE, 21, DEPARTMNT, 18

AGE VALUE DEPARTMNT IS INCORRECT

NEIGHBORING VALUES ARE 16 AND 16

ENTER "IGNORE," "HELP," OR NEW VALUE(s) – HELP

the designation DEPARMNT (a misspelling of DEPARTMENT) is neither a value (an age) for the category AGE nor another category name, so an error message is printed out. A request for help at this juncture would cause the subroutines VALHELP and SPCHELP to be called from the subroutine VALUE within the program VLOVL, and the following printout to be provided:

```
WOULD YOU LIKE TO KNOW THIS CATEGORY'S VALUES?–YES
THERE ARE 56 VALUES THAT CAN BE SPECIFIED FOR CATEGORY AGE
THE NUMBER FOLLOWING THE VALUE IS THE COUNT OF ELEMENTS IN THE
CATEGORY HAVING THE DISPLAYED VALUE
WOULD YOU LIKE TO HAVE ALL THE VALUES DISPLAYED–NO
HOW MANY VALUES WOULD YOU LIKE TO SEE DISPLAYED–10
HOW MANY VALUES WOULD YOU LIKE TO SKIP BEFORE THE FIRST ONE
IS DISPLAYED–0
HOW MANY VALUES WOULD YOU LIKE SKIPPED BEFORE THE NEXT VALUE
IS DISPLAYED–0
VALUES ARE:
  16          32
  17          96
  18         121
  19         135
  20         109
  21          92
  22         120
  23         112
  24         117
  25         114
WOULD YOU LIKE TO KNOW HOW TO SPECIFY THE VALUES
THAT DEFINE A CATEGORY–NO
ENTER "IGNORE" OR NEW VALUE(s)–20
    322 ELEMENTS WERE FOUND
    NEXT? ABORT
```

The next-to-last statement originates within the subroutine VALUE. It indicates the total number of employees who are 21, 20, or 18.

Example 3 – Use of the DISPLAY command to direct output to the batch printer:

The information

COMMAND?–GROUP, ROLLSTATUS, 1, SEX, 1, DEPARTMENT, 18

results in the production of a file of SSN's of all employees who are (1) on the rolls, (2) male, and (3) employed in Department 18. The entry

NEXT?–DISPLAY

causes control to be transferred to the DISPLAY program where the following dialog is initiated:

WOULD YOU LIKE THE DATA DISPLAYED HERE?–NO

WOULD YOU LIKE HEADINGS?–YES

ENTER DISPLAY TYPE–NAME

At this point, a job is automatically set up and submitted to the batch queue. Information written to the file TAPE3 includes an indication of the number of fields to be processed, the heading indicator, the display definition for each field, and the social security numbers (SSN) selected. The formulated batch job is submitted to a batch printer or a user terminal by program REPBTCH which uses parameters initialized in program BTCH to formulate a control card record. The batch run itself uses the program REPORT. After the job is run, output is produced in the following format, similar to that used at the TTY:

SERI	NAME
xxxx	xxx . . . xx
xxxx	xxx . . . xx
.	
.	
.	
xxxx	xxx . . . xx

The following instruction

NEXT?–ABORT

results in an exit from the DSOVL overlay, and a return to the command node.

Example 4 – How to define and use a special display format.

COMMAND?–ROLLSTATUS,1, DEPARTMENT, 18 SEX, 2

xxxx ELEMENTS WERE FOUND

yyy ELEMENTS WERE FOUND

zz ELEMENTS WERE FOUND

NEXT?–DISPLAY

WOULD YOU LIKE THE DATA DISPLAYED HERE?–YES

WOULD YOU LIKE HEADINGS?–YES

ENTER DISPLAY TYPE–DEFINE

The following statements indicate that the set of zz elements is to be displayed in a format to be defined by the user:

ENTER DATA FIELD(S) OR "STOP"–SS, NAME, TITLE

ENTER DATA FIELD(S) OR "STOP"–BIRTH-DATE, HOME-CITY, STOP

ENTER DISPLAY DEFINITION SAVE NAME–XYZ

The table produced under these specifications will have an entry in each field for each of the zz employees in the set, as indicated following:

	SS	NAME	T	BIRTH	HOME-CITY
(1)	x—9—z	x—28—x	x	xxxxx	x—15—x
.	.	.		.	.
.	.	.		.	.
.	.	.		.	.
(zz)	x—9—x	x—28—x	x	xxxxx	x—15—x

Example 5 – Use of the ELEMENT command:

COMMAND?–ELEMENT

WOULD YOU LIKE THE DATA DISPLAYED HERE?–YES

WOULD YOU LIKE HEADINGS?–YES

ENTER DISPLAY TYPE–NAME

ENTER ELEMENT NAME–xxxxxxxxxx

The "element name" requested is the social security number of the employee about whom information is to be furnished. The printout provided will appear as follows:

SERI NAME

xxxx x----28----x

New element names are entered, one at a time, in response to the request for an element name, until all names have been entered. The user then responds

ENTER ELEMENT NAME–ABORT

which will cause the printout

ELEMENT ABORTED

COMMAND?–

to be produced, and control to be returned to the command mode.

Example 6 – Use of the SORT command:

In this example, a data file is generated, saved, and subsequently used by the batch program SORTREP. The QUERY program SORT command causes batch output to be produced. The command input

COMMAND?–GROUP, ROLLSTATUS, 1, SEX, 2, DEPARTMENT, 18

causes the SSN's of all those women who are actively employed with Department 18 (CMLD) to be printed out. The entry

NEXT?–=

causes the SSN's of the selected employees to be saved. The entering of the word HI to the request

ENTER CATEGORY SAVE NAME–

indicates that the saved category is to be named HI. The following command entry

COMMAND?–SORT

causes control to be transferred to the SORT program of the QUERY system. The dialogue

CATEGORY TO BE SORTED?–HI

WOULD YOU LIKE HEADINGS ON THE INFORMATION–YES

ENTER DISPLAY TYPE–NAME

will cause the items in the HI category to be sorted and displayed under the column headings designated in the display format NAME which consists of the serial number and name of each employee. In the following exchange

ENTER SORT TYPE–ALPHA

the word ALPHA refers to a "built in" sort format which consists of one field only, NAME. The name ALPHA will cause the names of those employees whose SSN's were included in the designated category to be alphabetically arranged. All sorting and production of output is necessarily done in a batch job (program SORTREP) that has been set up by the programs BTCH and SRTBTCH. The output is disposed of as specified by the user in his answer to a query generated by the program SRTBTCH:

WOULD YOU LIKE YOUR OUTPUT SENT TO THE CENTRAL

SITE OR TO YOUR REMOTE ID? IF YOU PREFER

CENTRAL SITE, PLEASE TYPE IN "CENTRAL."

FOR REMOTE ID, TYPE IN YOUR THREE LETTER

ID DISPOSITION–

After the job has been submitted to the batch queue, control is returned to the command mode.

Example 7 – Use of the QUERY sorting facility:

First, a set of employees is generated at the terminal, the set then cataloged under the name ABC. Then another command is requested. The following exchange

COMMAND?–SORT, ABC, HEADING, DEFINE

indicates that the category ABC is to be sorted and the results printed out under the headings designated by a *defined* format (as opposed to a *built-in* format):

ENTER DATA FIELD(S) OR "STOP"–NAME, SS, BIRTH-DATE, GRADE, STOP

ENTER DISPLAY DEFINITION SAVE NAME–XYZ

The format defined, stored under the identifying name XYZ, consists of the employee name, SSN, birth-date, and grade. The following information is then entered:

ENTER SORT TYPE–DEFINE

ENTER SORT FIELD(S) OR "STOP"–BIRTH-DATE

ENTER "A" OR "D" FOR ASCENDING OR DESCENDING SORT–A

ENTER SORT FIELD(S) OR "STOP"–GRADE

ENTER "A" OR "D" FOR ASCENDING OR DESCENDING SORT–A

ENTER SORT FIELD(S) OR "STOP"–STOP

ENTER SORT DEFINITION SAVE NAME–CHI

A disk file is generated consisting of the display format, the sort format, and the social security numbers of the elements in the category. A batch job is set up with programs BTCH and SRTBTCH. The data file is submitted as input to the batch program, SORTREP, which will sort the SSN's and prepare a table of sorted data. These data are sorted first by birth-date, and then, within each birth-date, by grade and the results are printed out. The SORTREP output will take the following form (see Example (6)):

NAME	SS	BIRTH	GR
x - - - - - (28) - - - - - x	x - - - - - (9) - - - - - x	xxxxx	xx
.			
.			
.			
x - - - - - (28) - - - - - x	x - - - - - (9) - - - - - x	xxxxx	xx

When the sort has been completed, control is returned to the command mode.

Example 8 – Use of the COMPUTE directive:

Output produced as a result of this operation is always produced on-line (on the TTY printer). The category to be operated upon must have been previously defined. The QUERY instructions

COMMAND?–GROUP, ROLLSTATUS, 1, SEX, 1, DEPARTMENT, 18

XXXX ELEMENTS WERE FOUND

YYYY ELEMENTS WERE FOUND

ZZ ELEMENTS WERE FOUND

NEXT?–=

ENTER CATEGORY SAVE NAME–MEN

The preceding QUERY instructions cause the formation of a category made up of the SSN's of all male employees on the rolls of Department 18. The category is cataloged under the name MEN.

NEXT?–COMPUTE

ENTER NUMERIC CATEGORY NAME–GRADE

Computations performed on the category MEN will result in the following printouts:

NUMERIC CATEGORY GRADE HAS ZZ ELEMENTS

THE SUM OF THE VALUES OF THESE ELEMENT IS SSS

THE AVERAGE OF THESE VALUES OVER THE ELEMENTS IS AA.AAA

The average grade among male employees of Department 18 is AA.AAA. If n represents the number of employees (elements) in the set, and V_n indicates the value of a particular element, then the computed average is $(\sum_{1}^{n} V_n)/n$.

Example 9 – Use of the TABLE command:

The exchange

COMMAND?–TABLE

causes control to transfer to the TABLE program in the TBOVL overlay. The exchange

ENTER "HERE" OR "THERE" FOR TABLE OUTPUT DESTINATION–THERE

causes the output to be produced by a batch run. The responses provided to the following requests

ENTER CATEGORY NAME FOR ONE COMPONENT OF THE TABLE–SEX

ENTER CATEGORY NAME FOR ONE COMPONENT OF THE TABLE–GRADE

indicate that the table will denote two items of information, sex and grade.

The batch job is set up by the QUERY programs BTCH and TBLBTCH. Because the output is to be directed away from the TTY printer, the following dialog is initiated:

WOULD YOU LIKE YOUR OUTPUT SENT TO

THE CENTRAL SITE OR TO YOUR REMOTE ID?

IF YOU PREFER CENTRAL SITE, PLEASE

TYPE IN "CENTRAL." FOR REMOTE ID, TYPE IN

DISPOSITION–CENTRAL

The job is submitted to the batch queue from the TTY terminal and is executed by a control card record generated by TBLBTCH (the batch program is called TABLE). Control within QUERY is then returned to the Command Mode. The output takes the form indicated in the following table.

TABLE OF SEX VS GRADE

	SEX		
	1	2	TOTAL
GRADE			
00	a_{11}	a_{21}	Σa_{k1}
01	a_{21}	a_{22}	Σa_{k2}
02			
.			
.			
.			
18			
23			
61			
.			
.			
.			
64	a_{1n}	a_{2n}	Σa_{kn}
TOTAL	$\sum_{1}^{n} a_{1j}$	$\sum_{1}^{n} a_{2j}$	$\sum_{1}^{n} \sum_{1}^{2} a_{kj}$

ACKNOWLEDGMENTS

Credit is extended to Ms. S. Wybraniec for her many hours of labor in producing the flow charts for Appendixes A and E, and Ms. J. LaFlare for her assistance in debugging those portions of Programs SORTREP and INVERT which deal with the sorting of data.

APPENDIX A

FLOW DIAGRAMS OF THE QUERY SYSTEM

The diagrams that follow illustrate the flow of control through the programs and subroutines of the QUERY system. Names of routines and direct calling relationships between them are represented, although coding detail of individual routines has been omitted. The flowcharts of Appendix E provide greater detail.

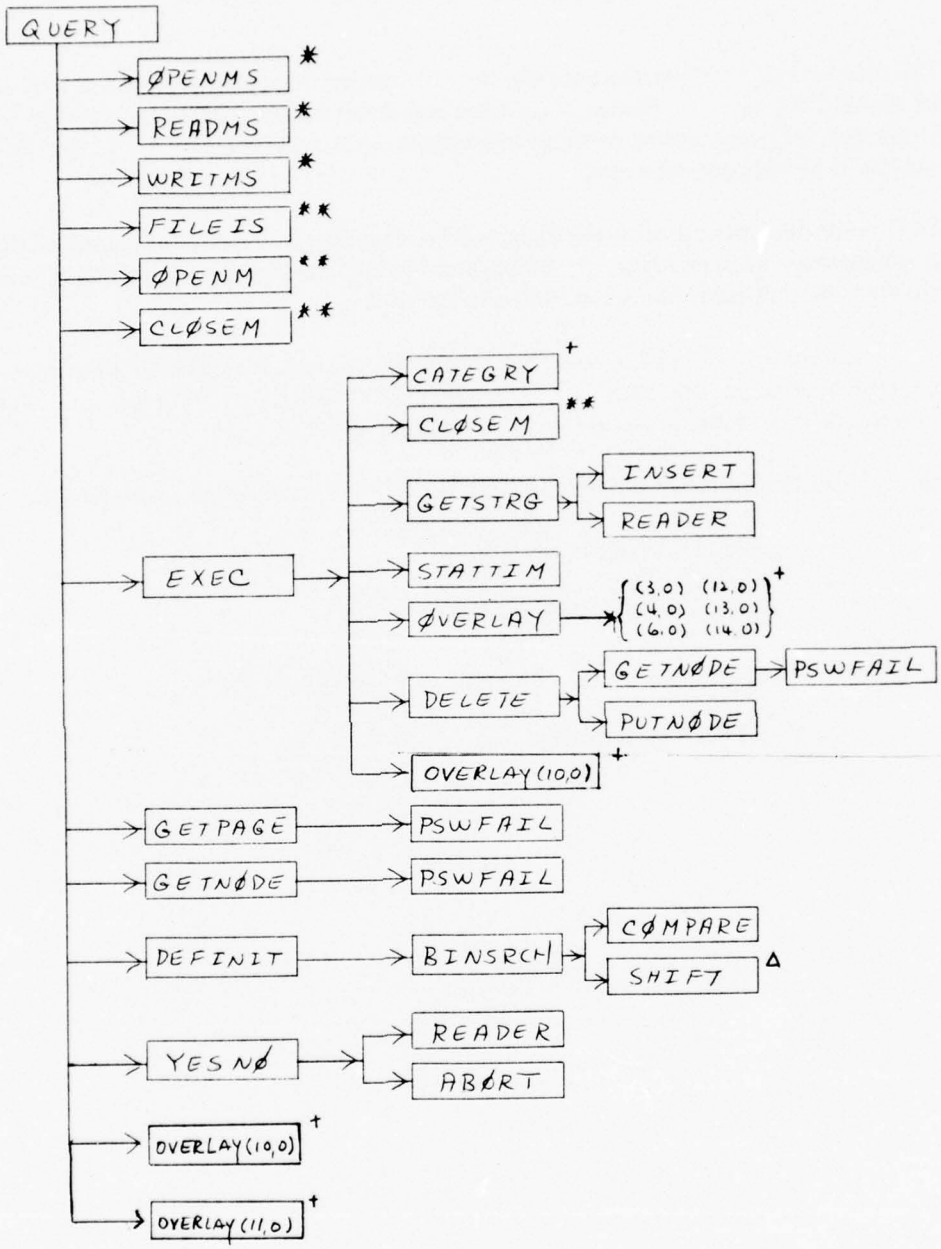
To simplify the chore of drawing these overlay diagrams, references to certain FORTRAN system subroutines—such as AND, OR, MOD, and SHIFT—used in the program have sometimes been omitted, even though they are included in the coding.

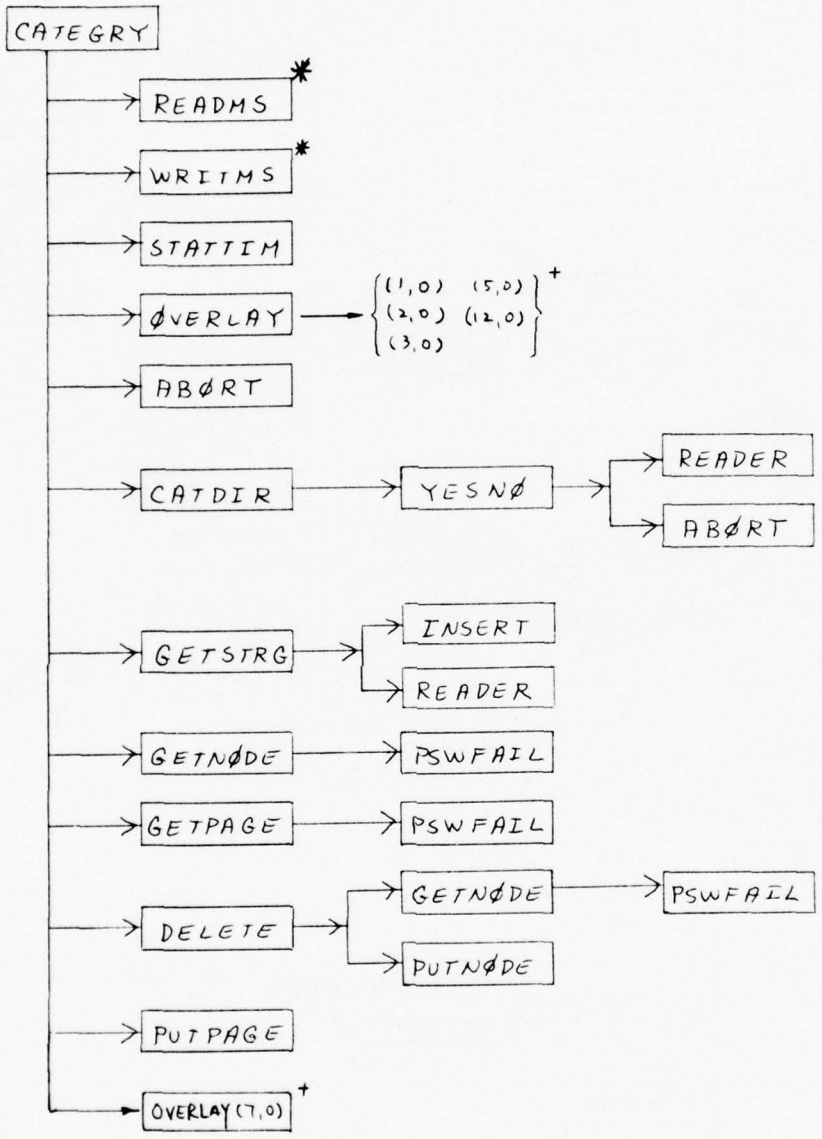
In the overlays (11,0) - (12,0), only the subroutines CHOICE and LOCF have been coded for the QUERY program; the others are taken from the COMRADE[2,3] library. Brief descriptions of some of these subroutines are included in Appendix C.

Superscripts used throughout the flow diagrams have the following connotations:

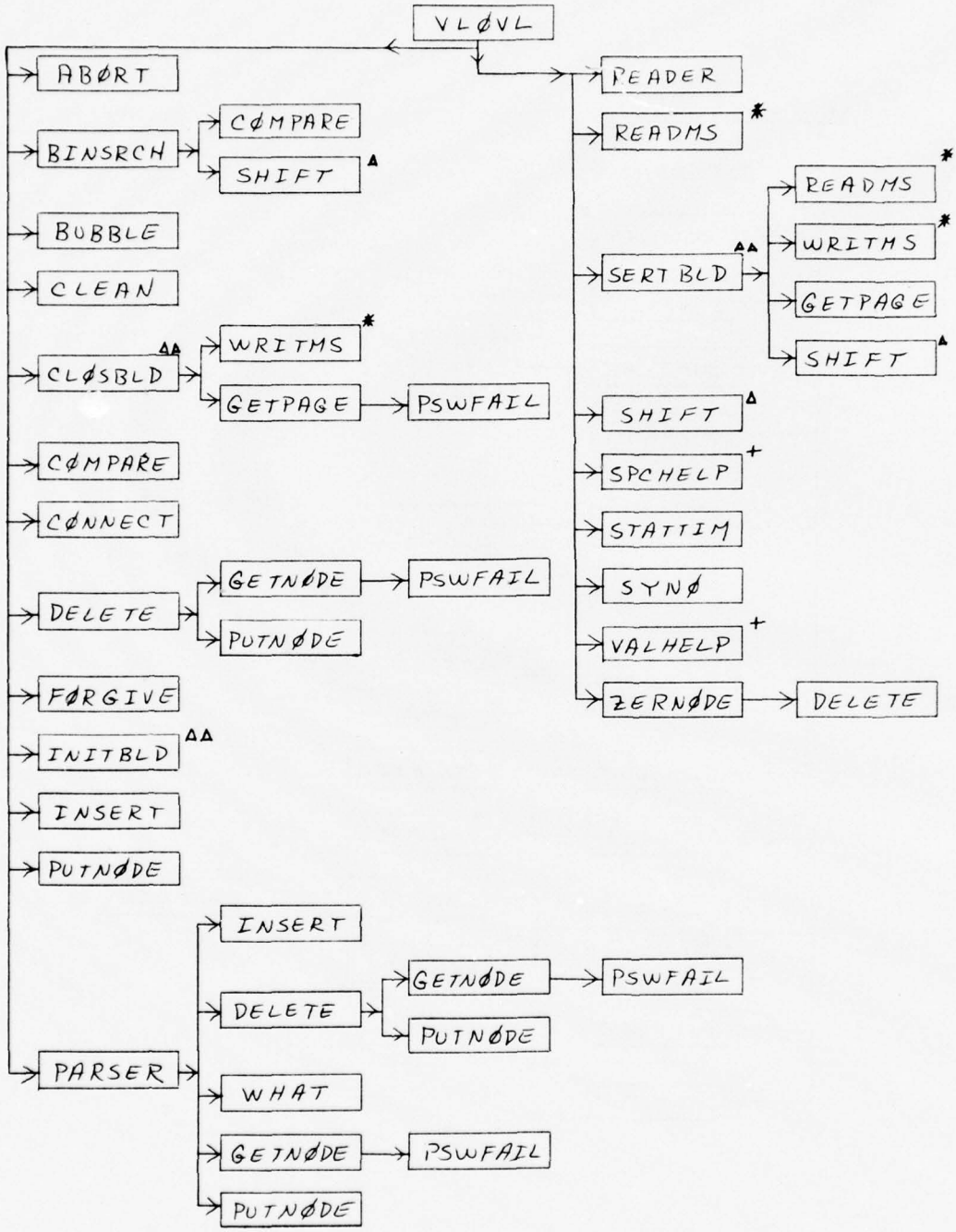
*	FORTRAN mass storage subroutine
**	Indexed sequential (IS) subroutine
+	Program or subroutine expanded on subsequent pages
△	FORTRAN system subroutine

PROGRAM QUERY
OVERLAY (0,0)

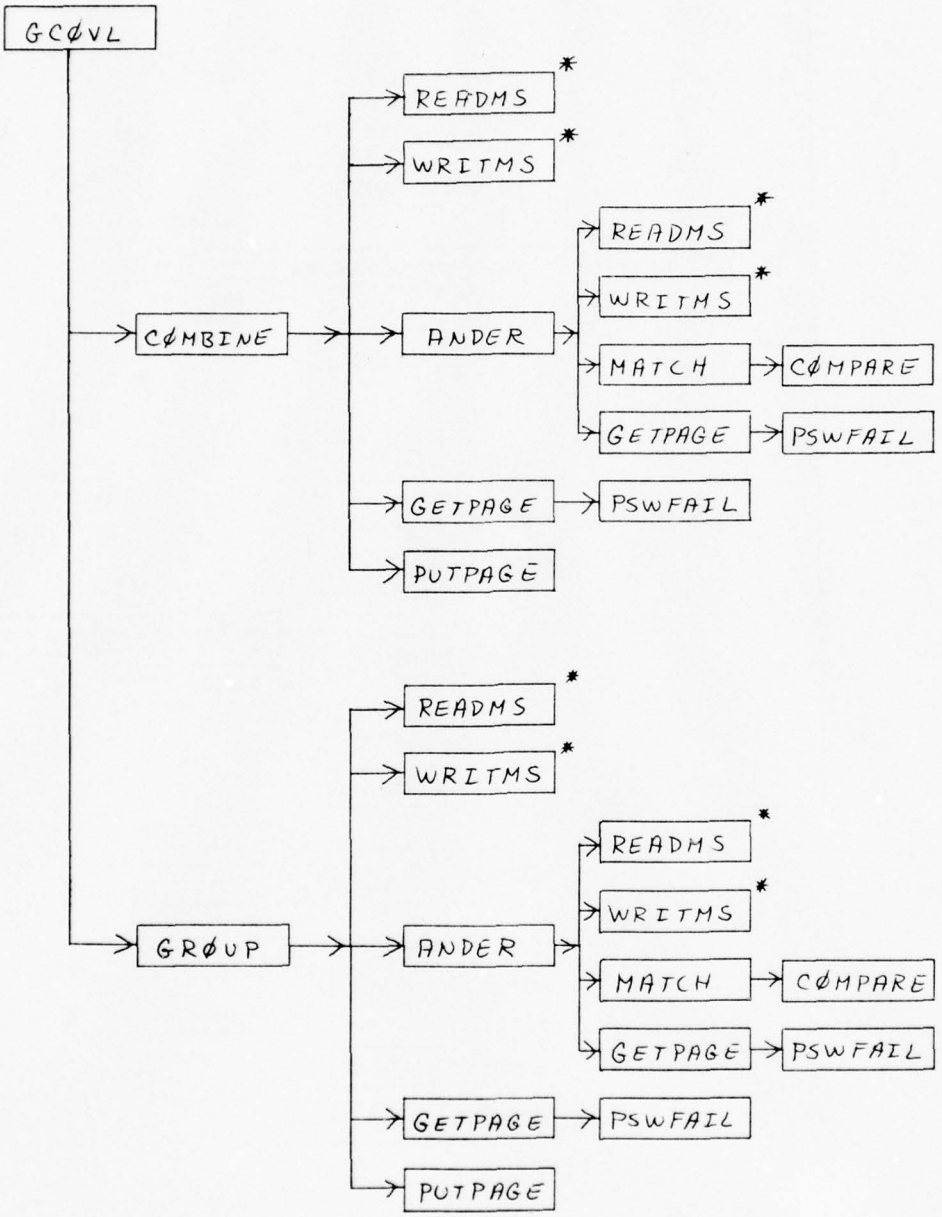



CATEGRY
READMS
WRITMS
STATTIM
ØVERLAY
(1,0) (5,0)
(2,0) (12,0)
(3,0)
+
ABØRT
CATDIR
YESNØ
READER
ABØRT
GETSTRG
INSERT
READER
GETNØDE
PSWFAIL
GETPAGE
PSWFAIL
DELETE
GETNØDE
PSWFAIL
PUTNØDE
PUTPAGE
OVERLAY(7,0)
+

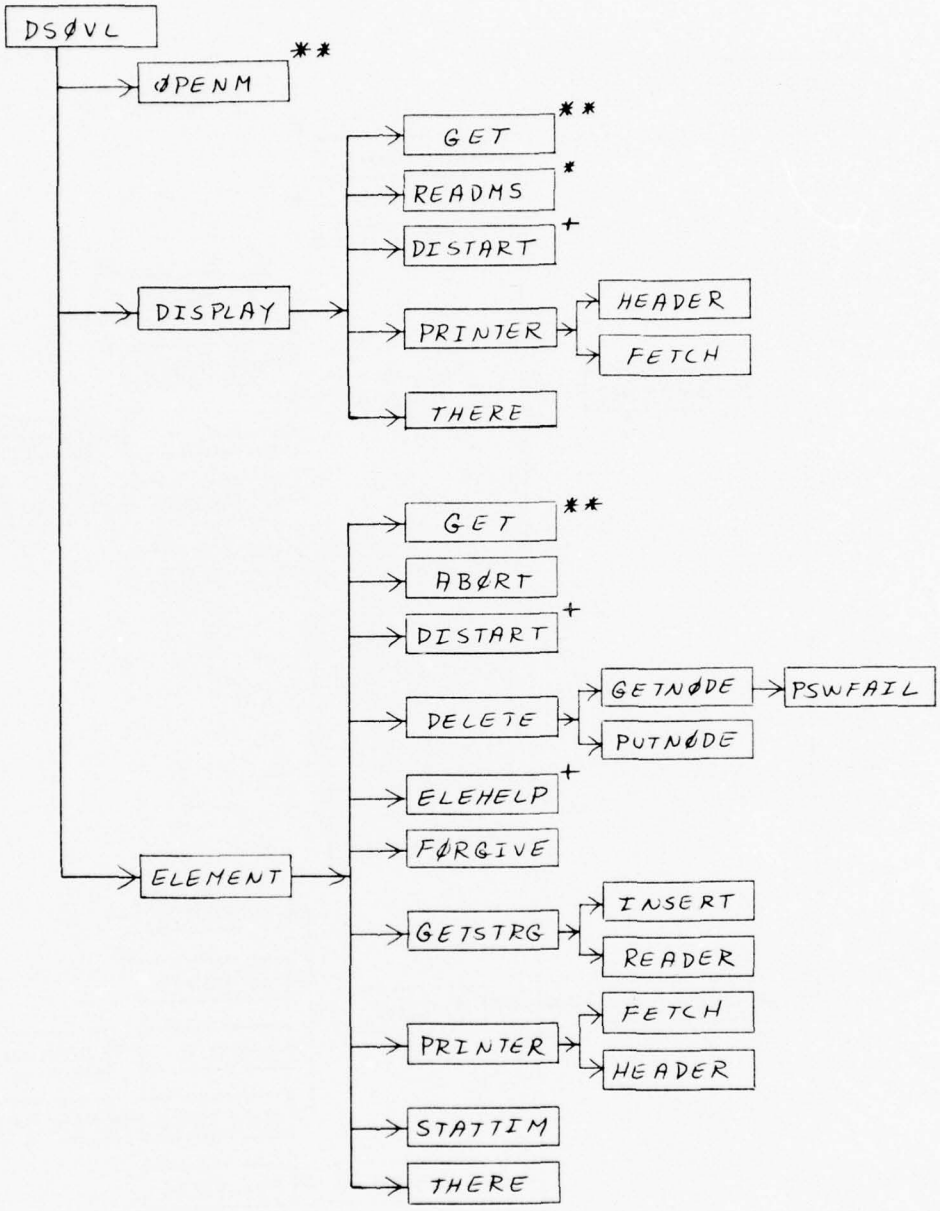
OVERLAY (1,0)


VLØVL
ABØRT
BINSRCH
CØMPARE
SHIFT
BUBBLE
CLEAN
CLØSBLD
WRITMS
GETPAGE
PSWFAIL
CØMPARE
CØNNECT
DELETE
GETNØDE
PSWFAIL
PUTNØDE
FØRGIVE
INITBLD
INSERT
PUTNØDE
PARSER
INSERT
DELETE
GETNØDE
PSWFAIL
PUTNØDE
WHAT
GETNØDE
PSWFAIL
PUTNØDE
READER
READMS
SERTBLD
READMS
WRITMS
GETPAGE
SHIFT
SHIFT
SPCHELP
STATTIM
SYNØ
VALHELP
ZERNØDE
DELETE

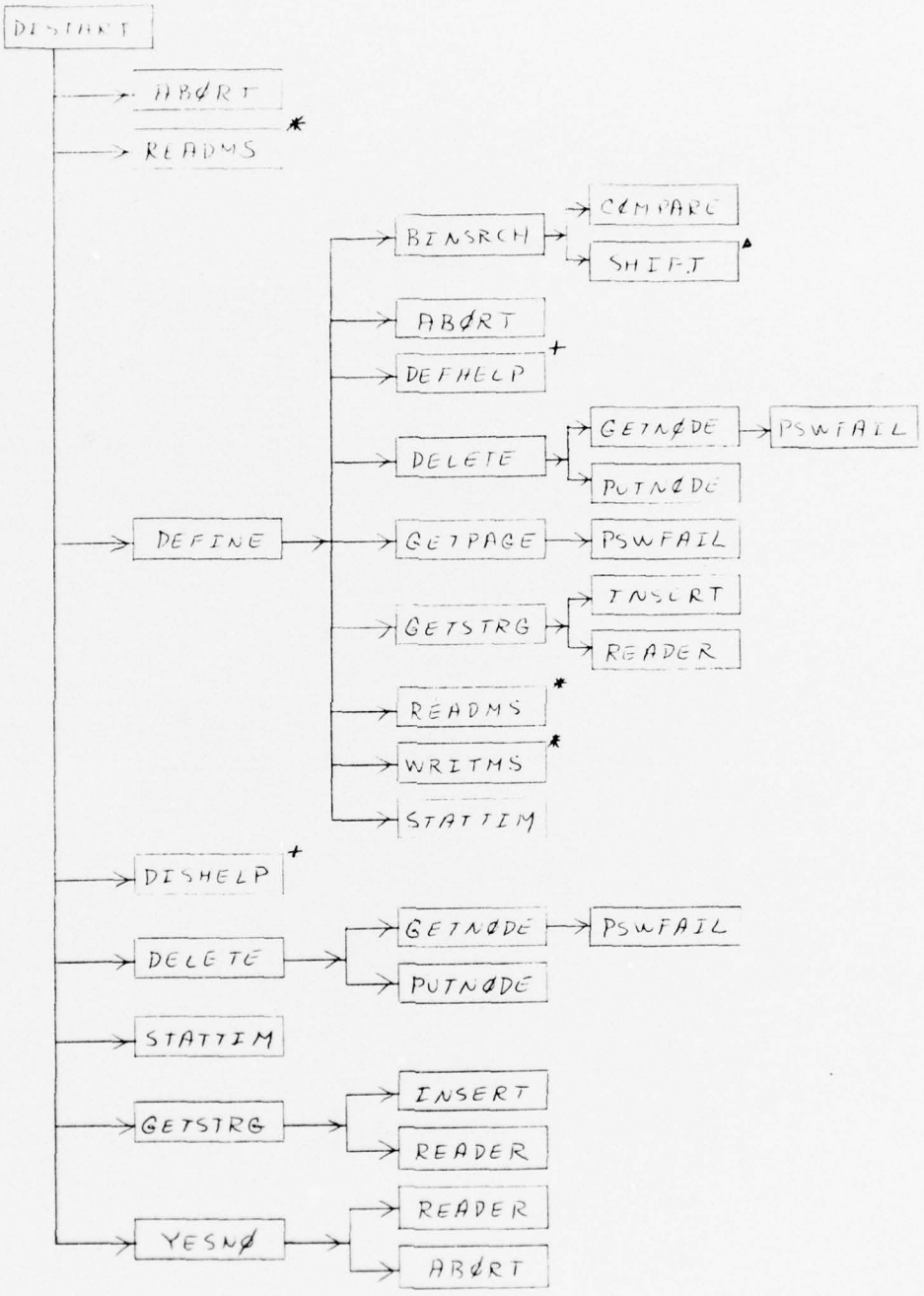
OVERLAY (2,0)


GCØVL
CØMBINE
READMS
WRITMS
ANDER
READMS
WRITMS
MATCH
CØMPARE
GETPAGE
PSWFAIL
GETPAGE
PSWFAIL
PUTPAGE
GRØUP
READMS
WRITMS
ANDER
READMS
WRITMS
MATCH
CØMPARE
GETPAGE
PSWFAIL
GETPAGE
PSWFAIL
PUTPAGE

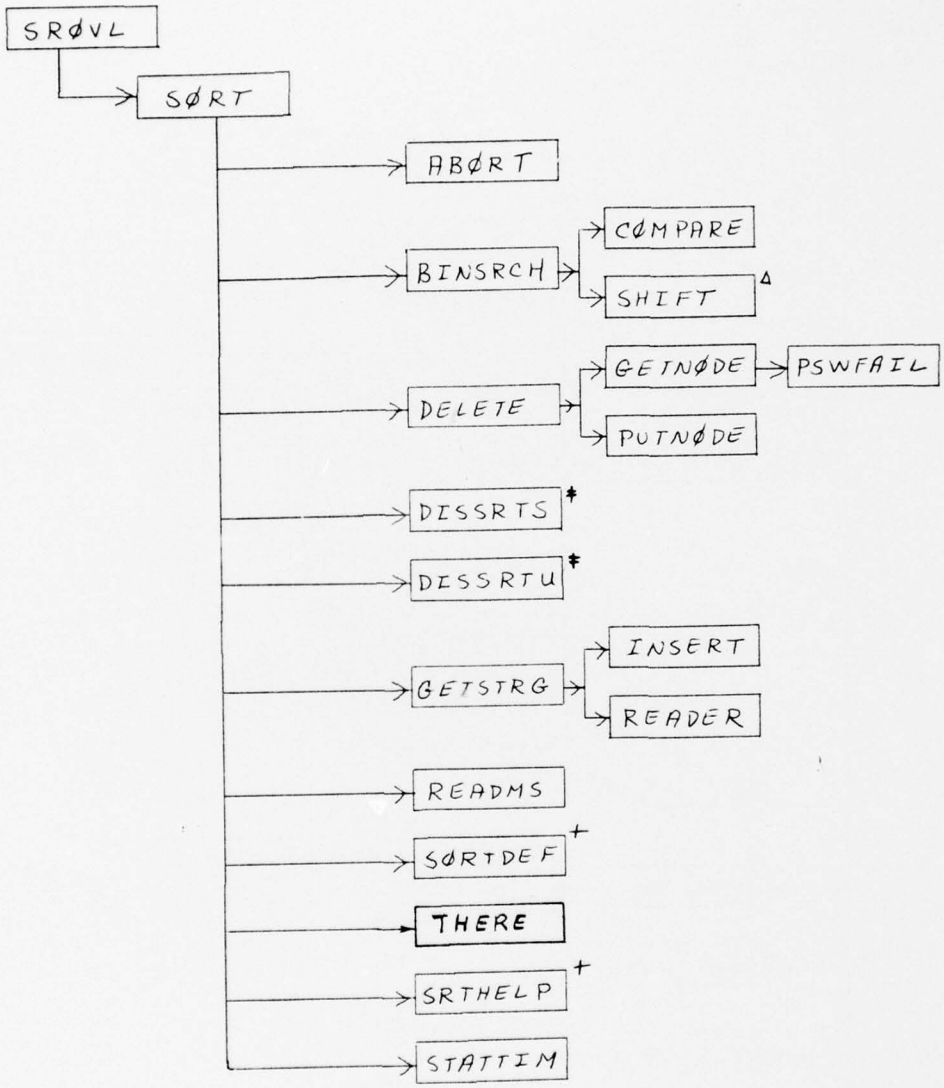
OVERLAY (3,0)


DSØVL
ØPENM
**
DISPLAY
GET
**
READMS
*
DISTART
+
PRINTER
HEADER
FETCH
THERE
ELEMENT
GET
**
ABØRT
DISTART
+
DELETE
GETNØDE
PSWFAIL
PUTNØDE
ELEHELP
+
FØRGIVE
GETSTRG
INSERT
READER
PRINTER
FETCH
HEADER
STATTIM
THERE

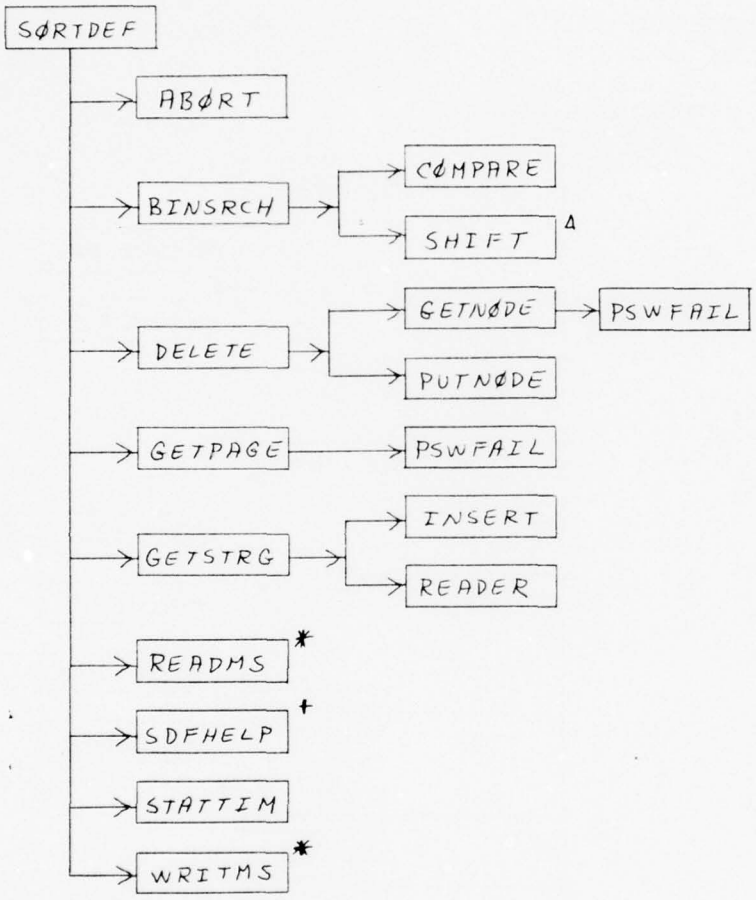
SUBROUTINE DISTART

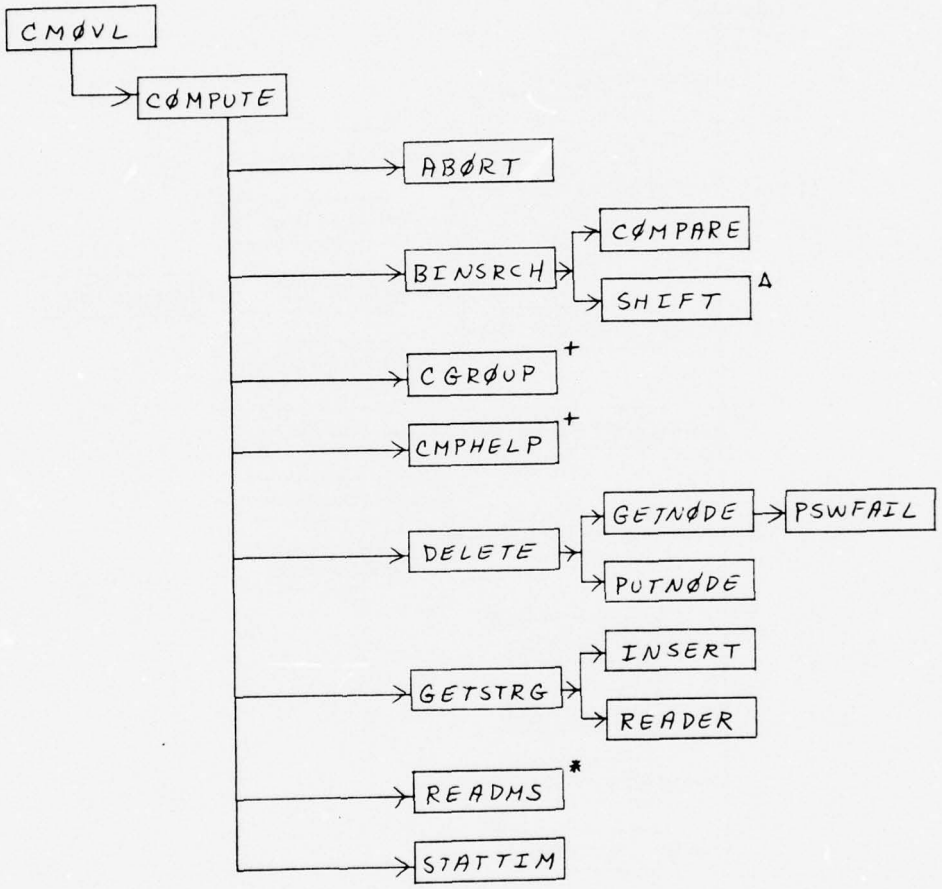


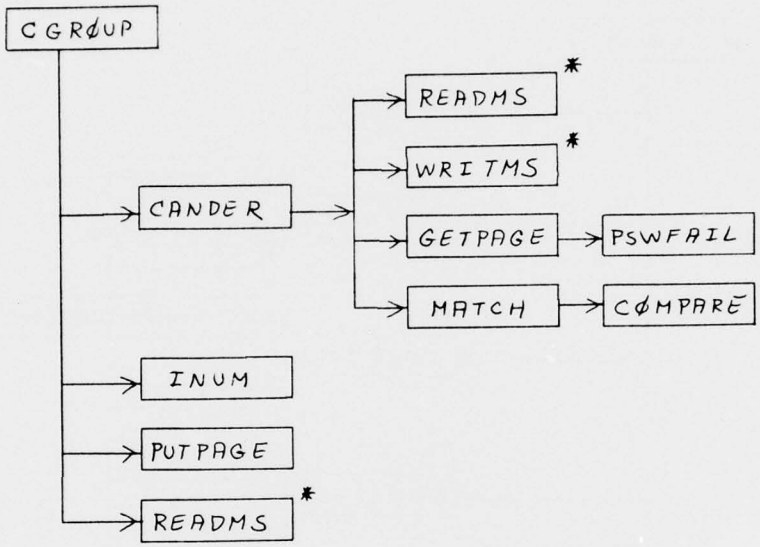
OVERLAY (4,0)

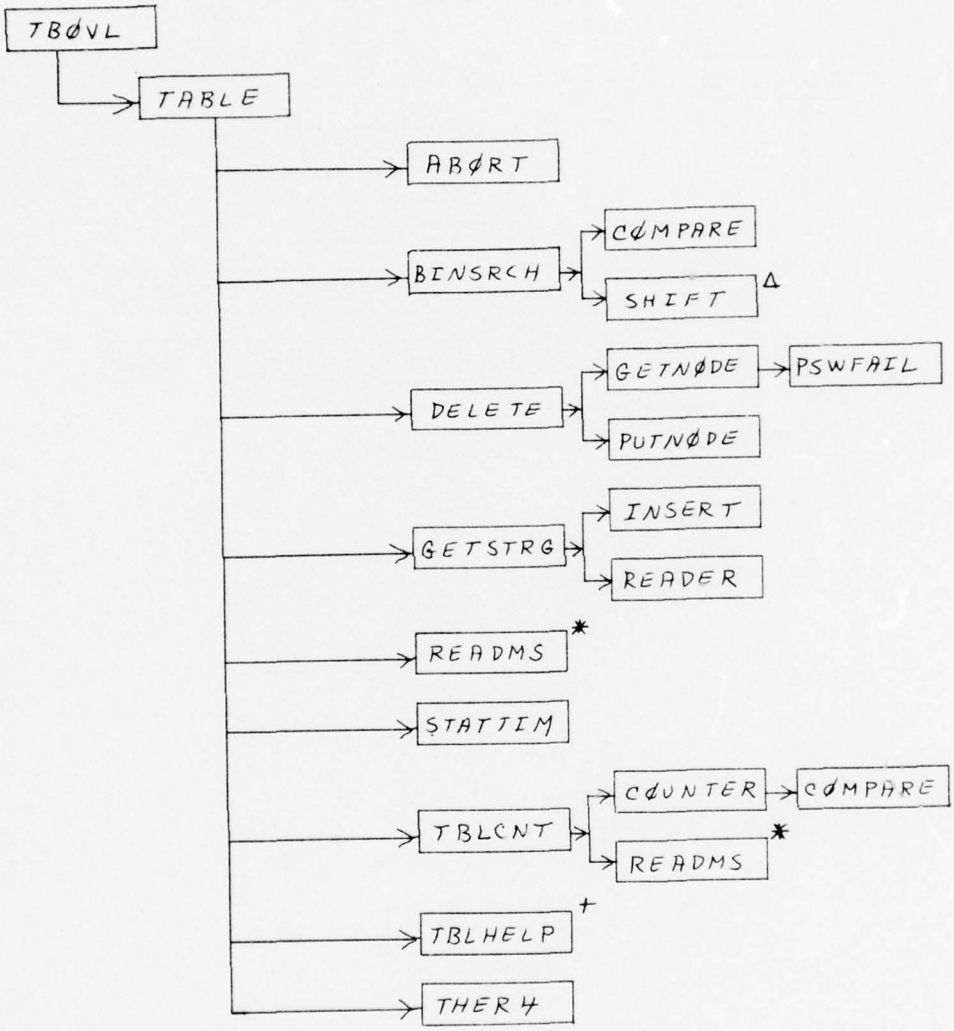

SRØVL
SØRT
ABØRT
BINSRCH
CØMPARE
SHIFT
DELETE
GETNØDE
PSWFAIL
PUTNØDE
DISSRTS
DISSRTU
GETSTRG
INSERT
READER
READMS
SØRTDEF
THERE
SRTHELP
STATTIM

SUBROUTINE SORTDEF

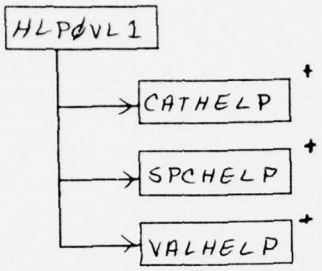



CMØVL
CØMPUTE
ABØRT
BINSRCH
CØMPARE
SHIFT
Δ
CGRØUP
+
CMPHELP
+
DELETE
GETNØDE
PSWFAIL
PUTNØDE
GETSTRG
INSERT
READER
READMS
*
STATTIM

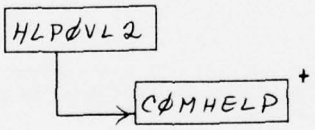

CGROUP
CANDER
READMS *
WRITMS *
GETPAGE
PSWFAIL
MATCH
COMPARE
INUM
PUTPAGE
READMS *

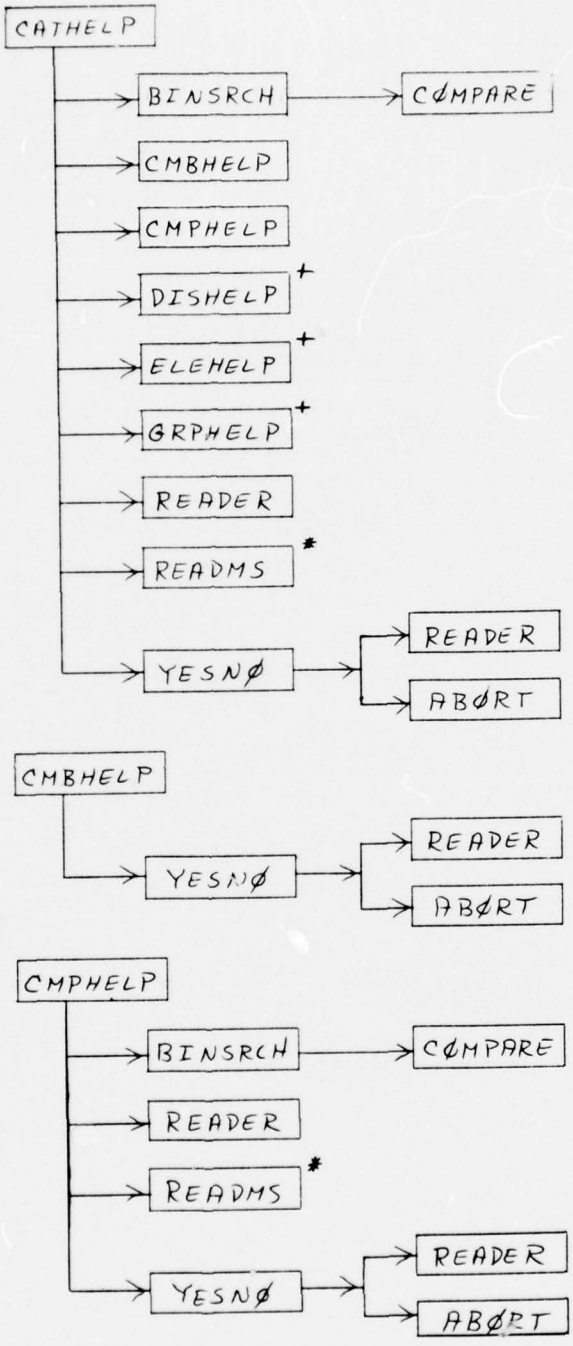

TBØVL
TABLE
ABØRT
BINSRCH
CØMPARE
SHIFT
DELETE
GETNØDE
PSWFAIL
PUTNØDE
GETSTRG
INSERT
READER
READMS
STATTIM
TBLCNT
CØUNTER
CØMPARE
READMS
TBLHELP
THER4

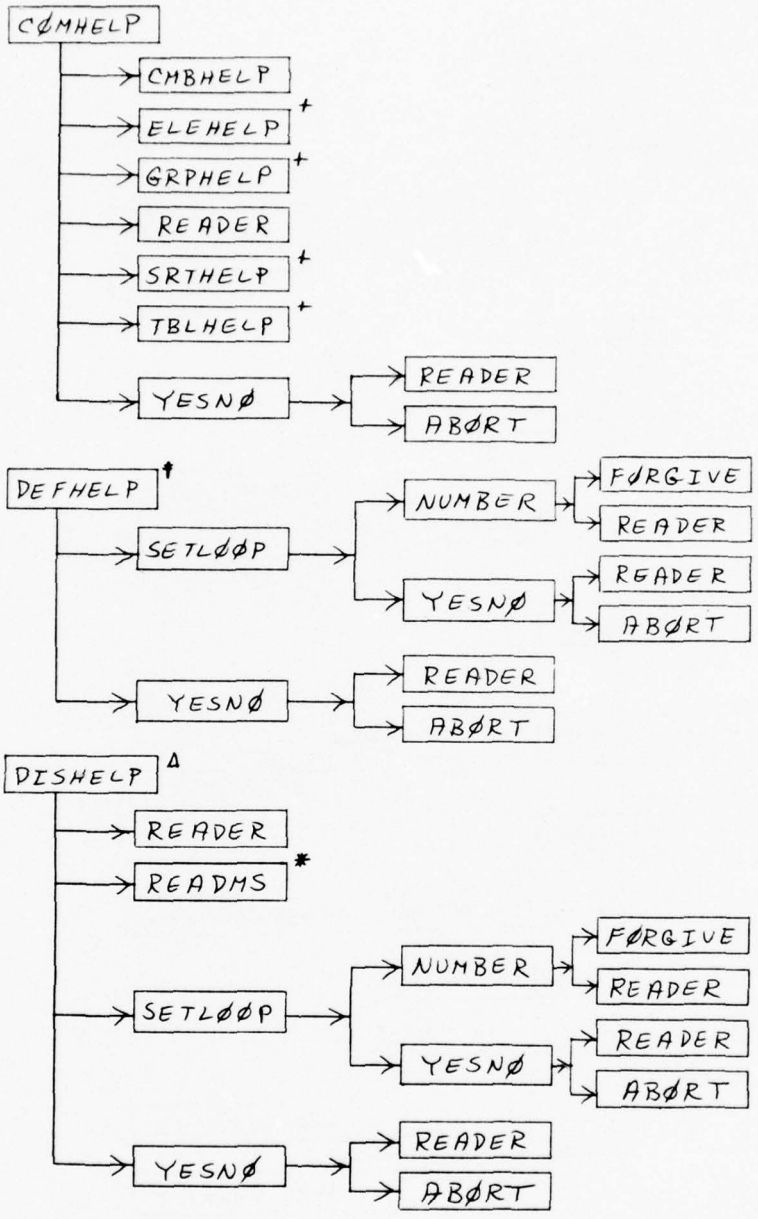
OVERLAY (7,0)

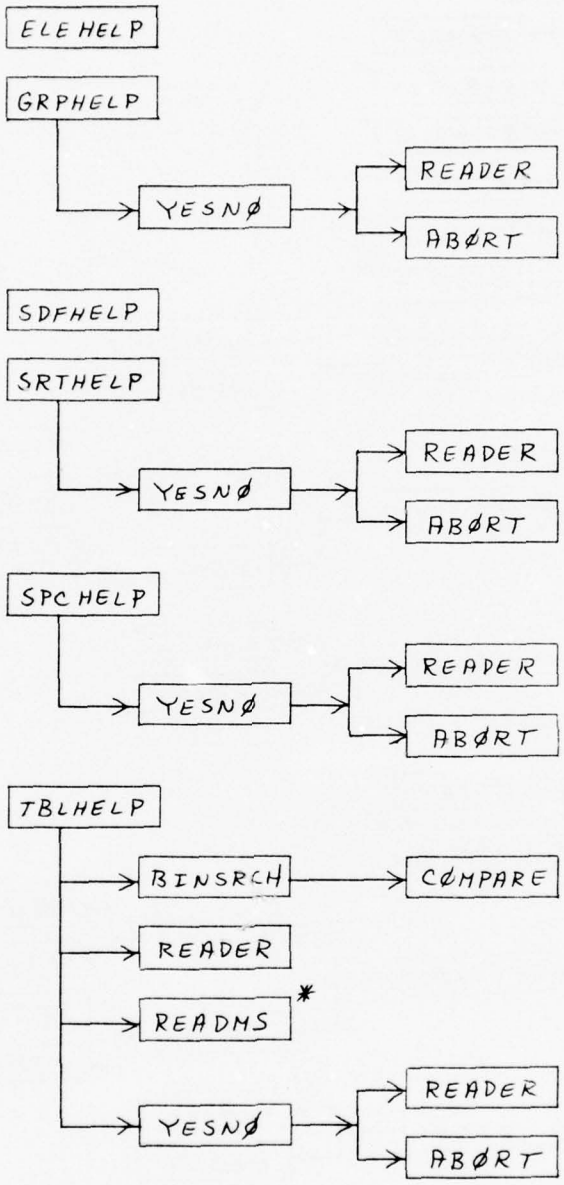


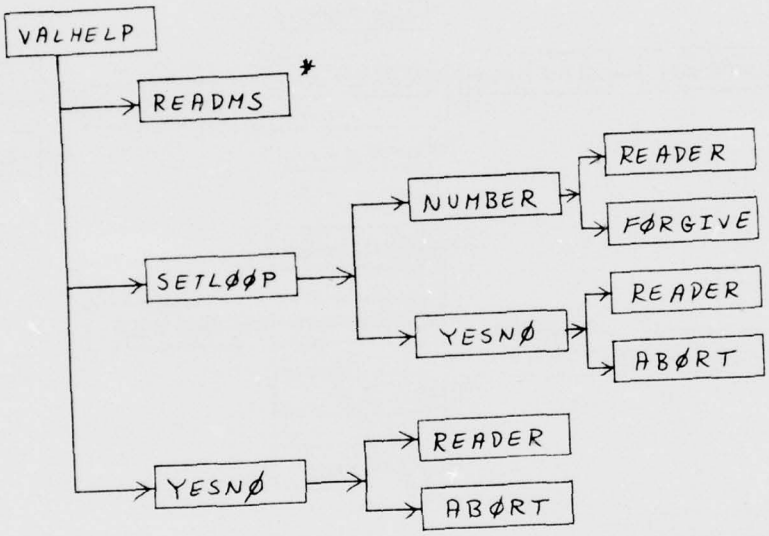
OVERLAY (10,0)

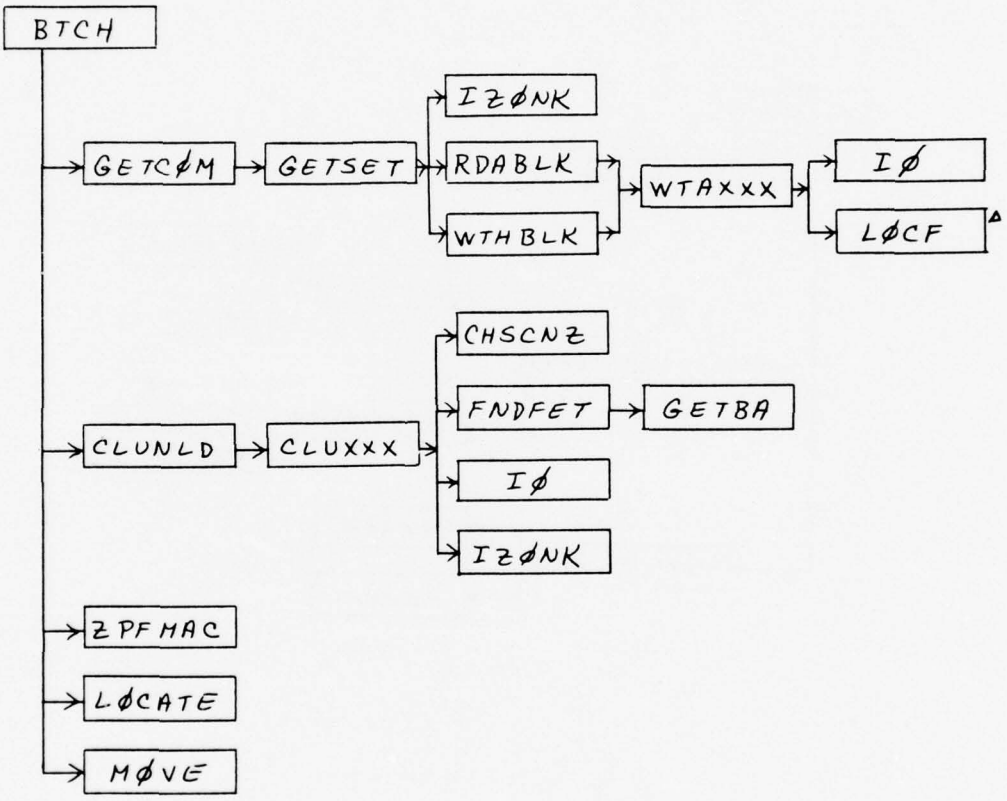


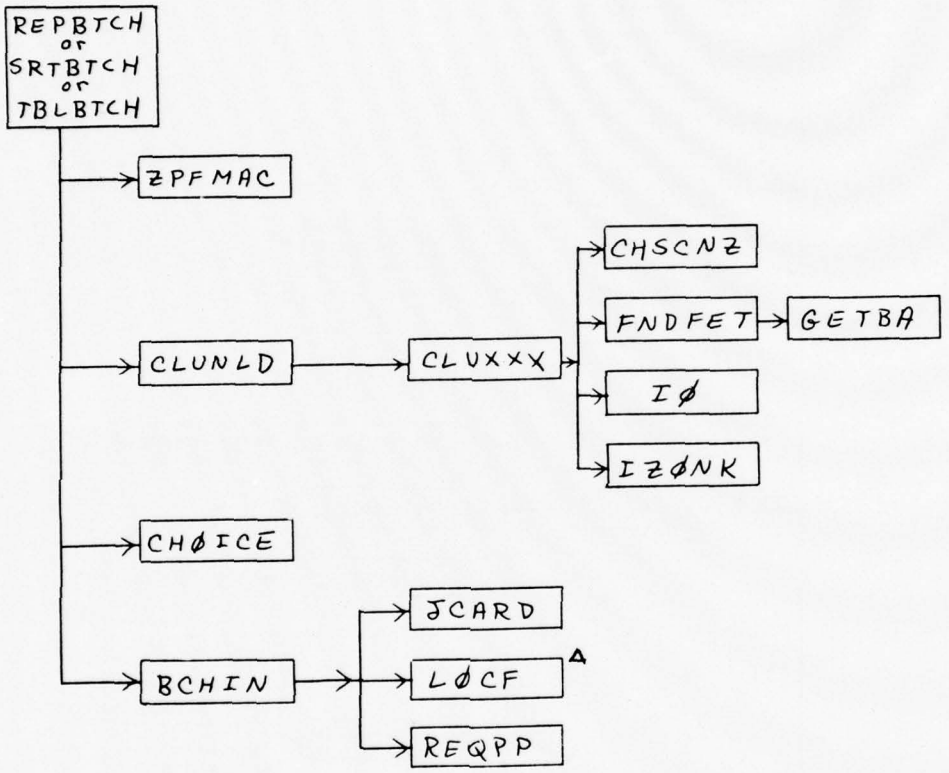

CATHELP
BINSRCH
COMPARE
CMBHELP
CMPHELP
DISHELP
+
ELEHELP
+
GRPHELP
+
READER
READMS
*
YESNO
READER
ABORT
CMBHELP
YESNO
READER
ABORT
CMPHELP
BINSRCH
COMPARE
READER
READMS
*
YESNO
READER
ABORT

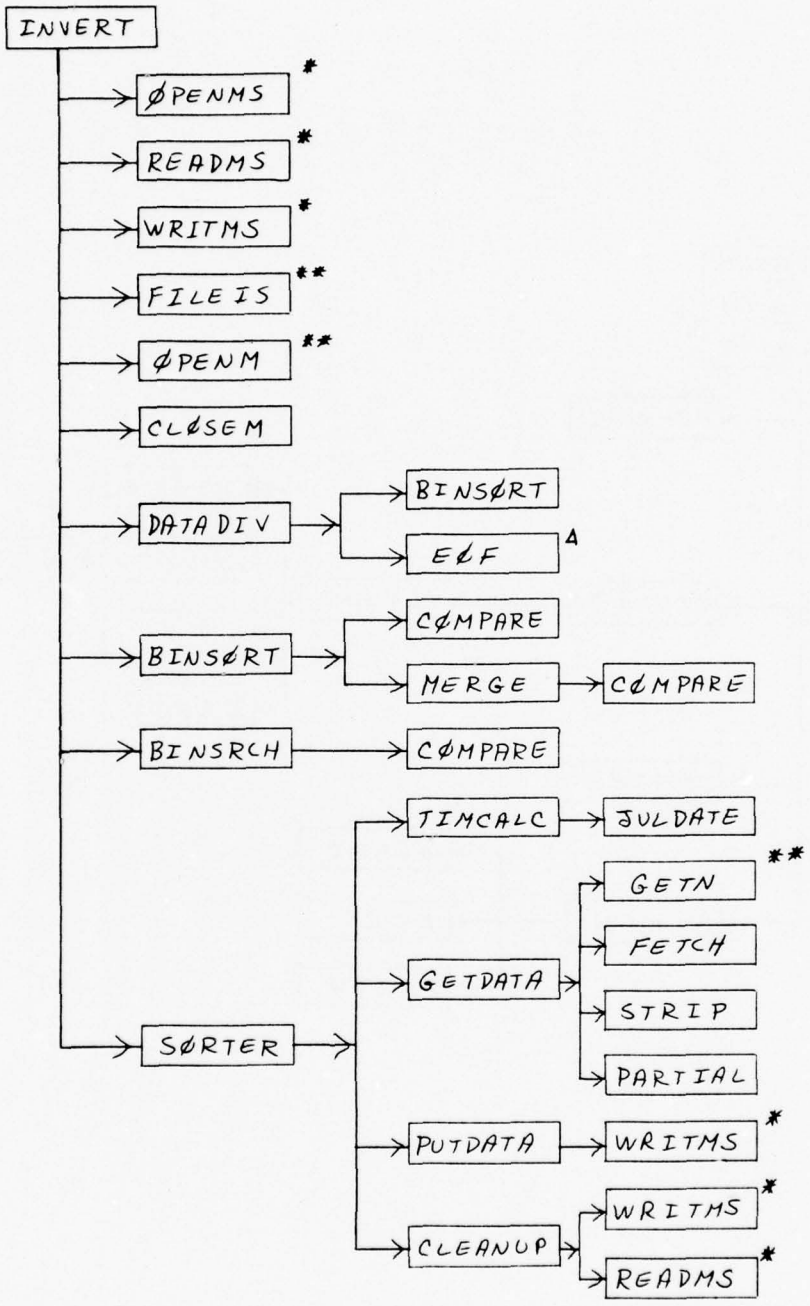

CØMHELP
CMBHELP
ELEHELP +
GRPHELP +
READER
SRTHELP +
TBLHELP +
YESNØ
READER
ABØRT
DEFHELP †
SETLØØP
NUMBER
FØRGIVE
READER
YESNØ
READER
ABØRT
YESNØ
READER
ABØRT
DISHELP Δ
READER
READMS *
SETLØØP
NUMBER
FØRGIVE
READER
YESNØ
READER
ABØRT
YESNØ
READER
ABØRT

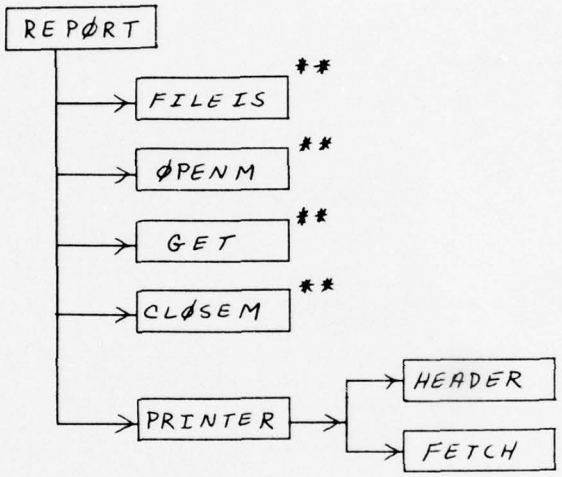

ELEHELP
GRPHELP
YESNØ
READER
ABØRT
SDFHELP
SRTHELP
YESNØ
READER
ABØRT
SPCHELP
YESNØ
READER
ABØRT
TBLHELP
BINSRCH
CØMPARE
READER
READMS
*
YESNØ
READER
ABØRT

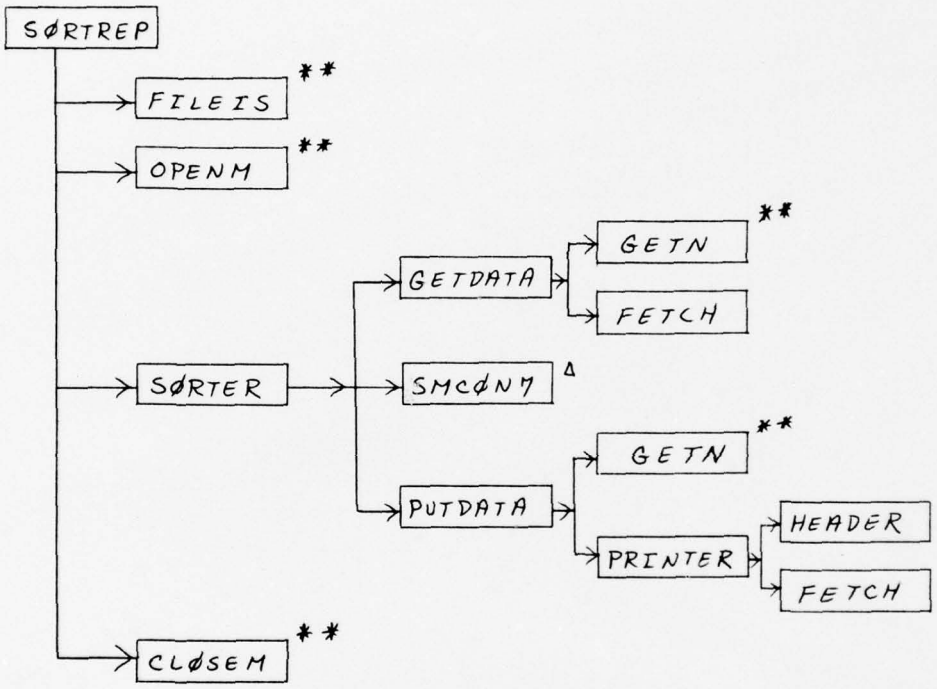

VALHELP
READMS
*
SETLØØP
NUMBER
READER
FØRGIVE
YESNØ
READER
ABØRT
YESNØ
READER
ABØRT


BTCH
GETCØM
GETSET
IZØNK
RDABLK
WTHBLK
WTAXXX
IØ
LØCF
CLUNLD
CLUXXX
CHSCNZ
FNDFET
GETBA
IØ
IZØNK
ZPFMAC
LØCATE
MØVE

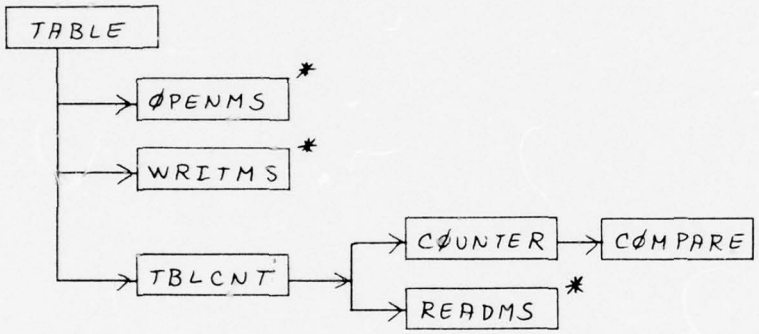

REPBTCH
or
SRTBTCH
or
TBLBTCH
ZPFMAC
CLUNLD
CLUXXX
CHSCNZ
FNDFET
GETBA
IØ
IZØNK
CHØICE
BCHIN
JCARD
LØCF
REQPP


INVERT
ØPENMS *
READMS *
WRITMS *
FILEIS **
ØPENM **
CLØSEM
DATADIV
BINSØRT
EØF
BINSØRT
CØMPARE
MERGE
CØMPARE
BINSRCH
CØMPARE
SØRTER
TIMCALC
JULDATE
GETDATA
GETN **
FETCH
STRIP
PARTIAL
PUTDATA
WRITMS *
CLEANUP
WRITMS *
READMS *


REPØRT
FILEIS
**
ØPENM
**
GET
**
CLØSEM
**
PRINTER
HEADER
FETCH


SØRTREP
FILEIS
**
OPENM
**
SØRTER
GETDATA
GETN
**
FETCH
SMCØNV
Δ
PUTDATA
GETN
**
PRINTER
HEADER
FETCH
CLØSEM
**

PROGRAM TABLE



APPENDIX B

DEFINITIONS OF COMMON VARIABLES

PROGRAM QUERY

NAMED BLOCK COMMON

BTCHDTA	
IFDB(7)	Constants retrieved from COMRADE storage and used as parameters on certain of the control cards used for a QUERY batch run.
COMMAND	
CSTRT	The starting node (or index) for a series of input words.
CURRENT	The index to the word of input being currently examined.
DISP	
DISDEF	Array which contains data for a maximum of 24 display definitions.
NUMDEF	Number of defined display formats.
HERE	Indicates whether output should be sent to the batch file (=3) or to the TTY (=4).
HEADING	Logical variable indicates whether a standard heading (=.T.) or no heading at all (=.F.) is desired on the printout.
NUMFLD	The number of elements in the record corresponding to DISDEF (1,I).
LINECNT	The number of lines generated for output by the DISPLAY program.
FREEAB	
FREEA	Specifies availability of core storage for category A.
FREEB	Specifies availability of core storage for category B.
FREEPG	
TOP	Number of the next available record.
GCV	
KBA	The key block for category A.
KBB	The key block for category B.
KBR	A temporary key block.
COUNT	The number of elements in a constructed category.
TYPE	A Hollerith variable which tells OVERLAY (3,0) to display data for one employee (=7HELEMENT) or more than one employee (=7HDISPLAY).

CATEG	A variable which holds the contents of the current node.
FILE	Number of a file which contains category data.
R,S	Indices used to reference data.
ERR	Error indicator; logical variable set to .TRUE. when no values entered.
IN	Record number of key block
GRPSTK	
STACKGR	Storage area (stack) for holding GROUP and COMBINE commands; as many as 16 may be entered.
STACKTP	Index which indicates a particular element of array STACKGR.
FIRST	A logical variable which indicates whether or not a category has been entered.
HLP	
ISUB	Indicator that specifies which of the HELP subroutines may be invoked at that point within the main overlay of QUERY.
NEWARR	
IARRAY	Used by the subroutine THERE for setting up control cards for a batch run.
READBUF*	
BRF	The array of element names in the predefined format, BRIEF.
NME	The array of element names in the predefined format, NAME.
DTL	The array of element names in the predefined format, DETAIL.
TITLE	The array of the names of the three predefined formats.
NUMTITLE	The array of the numbers of elements in each of the predefined formats.
IBF	Dummy array of 94 elements.

---

* In some subroutines the space occupied by this common block is used as core for storing data entered at the TTY.

SRT

SRTDEF	Array containing as many as 12 sort definitions.
NUMSDEF	The total number of sort definitions.
NUMSFLD	The number of elements in a sort definition.

TEMP

SWAPF	Array of integers used for keeping track of what records have been used for storage.

UNNAMED COMMON

Location (Octal)	QUERY Variable	Explanation
0 4	INODE (4) NODE (4,128)	Used in selecting the proper (next) entry from the input string.
1004	AVAIL	Contains the location of the next node.
1005 2005	PINDEX (512) TINDEX (513)	Indices to keep track of the mass storage records of two files.
3006	HEADA (2,256)	Core memory area for a header block for data of the category A.
4006	HEADB (2,256)	Core memory area for a header block for the data of the category B.
5006	TBLA (512)	Core memory area for a block of data of the category A.
6006	TBLB (512)	Core memory area for a block of data of the category B.
7006	CNTRL (3)	First data record (ID) read from mass storage; generated by program INVERT.
7011	KD (3,64)	Second mass storage data record; contains three key-block control words for each inverted category (a maximum of 64 such categories).
7311	FETARRY (35)	The indexed sequential file information table (FIT) in which information relating to file structure, status, and position is stored; every file has its own FIT.
7354	USERREC (63)	Core storage for a record from the IS file; every employee has his own "user record."
7453	TAG	The key to a user record, i.e., the social security number of an employee.

Notes:

(1) Record 2 of the inverted file contains three words of data for each inverted category (or 3*CNTRL (1) words, total). This array, KD, read from disk by program QUERY, is computed in program INVERT as array KCB.

(2) Record 3 of the inverted file, the array of available formats (5*CNTRL (2) words), is read from disk into array HEADB by program QUERY. Core storage beginning with HEADB is called FD (5,307) in subroutine DEFINIT. In the program INVERT the formats are called FMT.

(3) Formats are defined in the subroutine DEFINIT and placed in array FDDEF (5,100) after being retrieved from array FD. Array FDDEF is equivalent to array DEF in DEFINIT, and DEF, in turn, is equivalent to array HEADA. There are approximately 140 formats in FD available for definition and use in QUERY.

PROGRAM REPORT

NAMED BLOCK COMMON

DISP

DISDEF (3,24)	Display formats.
NUMDEF	Number of display definitions in run.
HERE	Destination of output.
HEADING	Specifies whether or not headings will appear on output.
NUMFLD	Number of elements in record corresponding to DISDEF (1,I).
LINECNT	Number of lines to be printed per page.

READBUF

LINEOUT (137)	A line of output to be generated, one character per word.
FIELD (137)	Data selected from the user array by means of information contained in the format array.

UNNAMED COMMON

Location (Octal)	REPORT Variable	Explanation
0	FDDEF (5,102)	The array of defined formats.
776	FETARRY (22)	The indexed sequential file information table (FIT); it stores data relative to file structure, status, and position.
1024	USERREC (63)	Core storage for a record from the IS file.
1123	BUFLOC (1300)	Buffer area used by the IS subroutines.
3547	TAG	The key to a user record, i.e., the social security number of an employee.
3550	ERRCODE	Error code.

NAMED BLOCK COMMON

ARGS

IDUM (2)	Words used by the sort/merge program.
IN (100)	Area for the input data.
EOF	End-of-file indicator, set at the end of the IS file information.

C

FDDEF (5,102)	Display definitions.
FETARRY (22)	File Information Table for IS.
USERREC (63)	An employee record.
BUFLOC (1300)	IS Buffer.
TAG	Key location.
ERRCODE	Variable to receive status code from IS.
SDDEF (5,100)	Sort definitions.
TOTSIZE	Total size in characters of sort field(s).
WRDSIZE	Number of words represented by TOTSIZE.

DISP

DISDEF (3,24)	Display formats.
NUMDEF	Number of display definitions in this run.
HERE	Specifies destination of output.
HEADING	Specifies whether or not headings will appear on output.
NUMFLD	Number of elements in record corresponding to DISDEF (1,I).
LINECNT	Number of lines put out on page.

SRT

SRTDEF (3,12)	Sort formats.
NUMSDEF	Number of sort definitions.
NUMSFLD	Number of elements in record corresponding to SRTDEF (1,I).

PROGRAM TABLE

NAMED BLOCK COMMON

READBUF (occurs in Subroutine TBLCNT)

SUBTOT (21)	Sums of elements having specific header values in each of two categories.
HEADT (2,21)	The header values for two categories.
CNT (20)	(Not used)
LCNT	The sum of subtotals across one row of the table.
TTL (20)	The sum of subtotals down one column of the table.
SUM	The sum of TTL across a row.

UNNAMED COMMON

Location (Octal)	TABLE Variable	Explanation
0	HEADA (2,256)	Core memory area for a header block for data of the category A.
1000	HEADB (2,256)	Core memory area for a header block for data of the category B.
2000	TBLA (512)	Memory space for a block of data of the category A.
3000	TBLB (512)	Memory space for a block of data of the category B.
4000	INDEX (64)	Array to keep account of the records of a mass storage file.
4100	KB (64,2)	The key blocks for two categories.
4300	CATEG (2)	The names of two categories.

PROGRAM INVERT

NAMED BLOCK COMMON

ARGS

IDUM (2)	Words used by the sort/merge program.
IN (100)	Area for the input data.
EOF	End-of-file indicator, set at the end of the IS file information.

INIT	
MULT (3)	Values extracted from the fifth word of the display format (≡1).
DISP (3)	Three of six values extracted from the fifth word of the format (≡0).
FIRST	Indicates that first employee (only) will not be checked for value of MUL.
FOR (5)	The display format for a specific field of data from USERREC, an employee's record.
PARAM	
TIME	Indicates whether or not a date is to be converted to an inclusive number of years.
VARSCAN	Indicates whether or not the first word of an extracted field contains blanks.
FLDLEN	The total length of field in the user record.
SCHAR	The starting character, within the field of the sub-field indicated by QUERYNM.
LENS	Length of subfield.
SPACE	Indicates whether or not field is vacant.
COBNAME	Name of the field in the COBOL input.
POINTER	
P	Indicates the number of 2-word entries in header block, i.e., two words per employee. ($\leq$256)
PTR	Indicates the number of SSN in the data block being formed.
SIS	
FETARRY (22)	File Information Table for IS.
USERREC (63)	An employee record.
BUFLOC (1293)	IS buffer.
TAG	Key location.
ERRCODE	Variable to receive status code from IS.
BUFSIZE	Size of IS buffer.
STRUCT	
HEAD (2,257)	Memory storage for header values as blocks are built up.
TBL (512)	Memory storage for blocks of SSN.
KCB (3,32)	Memory storage for key control blocks.
KB (64)	Memory area for the key block.

KEY	The number of a particular category in the inverted file.
IN	Record number for writing key block

UNNAMED COMMON

Location (Octal)	INVERT Variable	Explanation
0	FMT (5,256)	The formats defined for each field in the employee data file. The maximum is 256 of these five-word formats.
2400	SOFTBUF (10000)	Sort buffer.

Location (Octal)	DATADIV Variable	Explanation
0	IDUM (3000)	Dummy Storage.
5670	LEVEL (250)	Type of input data record from COBOL output.
6262	STACK (250)	Indicator of the current data record being examined.

Location (Octal)	MERGE Variable	Explanation
0	IFMT (5,256)	The formats defined for each field in the employee data file.
2400	TEMP (1280)	Storage used when merging the formats into order.

APPENDIX C

BRIEF DESCRIPTIONS OF PROGRAMS AND SUBROUTINES OF THE QUERY SYSTEM

THE (0,0) OVERLAY (QUERY)

QUERY* — The main program of the (0,0) overlay. If performs the following tasks:

1) Initializes common variables, FORTRAN mass storage subroutines, and IS subroutines.
2) Reads three data records from Disk 1.
3) Sets up three display formats.
4) Sets up one sort format.
5) Accesses a tutorial subroutine, COMHELP.
6) Calls EXEC, the executive subroutine of the system.

BINSRCH — Finds the index of an element in a given array. That element has a value equal to the specified input value.

CATDIR — Determines the nature of an entry in the CATEGRY subroutine.

CATEGRY* — Subroutine called from EXEC when either of the commands GROUP or COMBINE has been entered. It performs the following functions:

1) Gets next entry from stack.
2) Determines nature of entry (command, directive, or value) and transfers to appropriate code.
3) If values are to be entered, calls in OVERLAY (1,0) to read them. If appropriate, transfers to OVERLAY (2,0) and performs a GROUP or COMBINE operation on the designated set of elements.
4) If a DISPLAY, COMPUTE, or ELEMENT directive has been entered, calls in the appropriate overlay.
5) If a TIME, ABORT, HELP, STOP,), or = directive has been entered, goes to the appropriate code.

COMPARE — Compares two input numbers to determine the relationship of one to the other (greater than, equal to, or less than).

DEFINIT* — Builds up an array with definitions (formats) of designated fields, five words per field.

---

*A flow chart of corresponding code is provided in Appendix E.

DELETE	Deletes an entry (element) from the input string; releases a node.
EXEC*	The executive subroutine of the QUERY system. It performs the following functions: 1) Reads and interprets a command -- permissible ones are GROUP, COMBINE, SORT, STOP, ELEMENT, TABLE, TIME, HELP. 2) Transfers to appropriate subroutine or overlay.
FORGIVE	Insures that a BCD number has a specified number of numeric digits.
GETNODE	Gets the next location (node) in the data structure available for input elements.
GETPAGE	Gets a new record number for storing data of QUERY.
GETSTRG	Fetches from the TTY terminal buffer a string of entries, consisting of commands and data.
INSERT	Updates the input array of indices and adds an entry to the input string. INSERT is an entry point of DELETE.
PUTNODE	Releases an input element location (node) from the data structure.
PUTPAGE	Releases a record and makes its name available.
READER	Reads a complete line of input transmitted from the TTY.
STATTIM	Determines elapsed central processor time and "connect" time.
YESNO	Determines which of the two entries Yes or No has been made by the TTY user.

All subroutines which offer help on the use of QUERY are described in the section entitled "On-Line Information About QUERY Commands and Directives."

THE (1,0) OVERLAY (VLOVL)

VLOVL* The main program of the (1,0) overlay:

1) Determines whether or not the current input category is the first category to be read in.
2) If so, reads key block from mass storage into KBA, otherwise, into KBB.

BUBBLE Removes duplicate data from input string.

BUILD* Builds a new data file (TBLB) for a certain category from the data contained in another data file (TBLA). The elements to be selected from TBLA are identified by data specified in the TTY input.

CLEAN Removes redundancies from the input list of delimiting values and connecting words.

CONNECT Checks for connecting words, and replaces all such words with the word CON. The output from the routine CONNECT is used as input to PARSER.

PARSER Separates the input string into individual values and connectors, and stores these words in the array NODE.

SYNO Checks for synonymous words which specify the starting and ending values of a range of values.

VALUE* Gets the values associated with a category as indicated by the input list:

1) Determines whether or not the next entry lies within the current record.
2) Checks for the presence of special connecting or delimiting words.
3) Calls on the subroutine VALHELP to print information about entering data via the TTY.
4) Separates the input string into units and stores the values contiguously in an array.
5) Checks the string for duplicate and zero values.
6) Initializes and enters the subroutine BUILD to construct a file of pertinent elements.
7) Closes the file being built and completes a key block for it.

WHAT — Searches the input string for the words CON and NOT.

ZERNODE — Removes zero-valued nodes from the input string.

THE (2,0) OVERLAY (GCOVL)

GCOVL* — The main program of the (2,0) overlay. Determines which operation is to be performed and transfers to the appropriate subroutine.

ANDER — Writes out a series of records which are pointed to by the words KBR_j. These records contain certain data chosen from records TBLB, by means of a test performed by subroutine MATCH. In the test, each of the records pointed to by KBB_j is compared with all records specified by KBA_i. Any match is indicated by ORing a 1 into the last bit position of the element of KBB_j.

COMBINE — Performs the COMBINE operation. Constructs records which consist of the OR'd data of two input sets. The new records combine data from one set with all those elements of the other that are different.

GROUP — Called to GROUP data. Constructs records $TBLA_i$ which contain only those elements of $TBLB_j$ which match some element of the set $TBLA_i$. This is the operation of ANDing two sets or records.

MATCH — Compares elements of category B with a given header value of category A with elements of category B of all header values. Indicates a match in the words of category B.

THE (3,0) OVERLAY (DSOVL)

DSOVL* — The main program of the (3,0) overlay. It determines which type of display is to be produced, and transfers control to DISPLAY (set of employees) or ELEMENT (one element).

DEFINE* — Defines and saves a new format for use in addition to those predefined formats, in printing output.

DISPLAY* — Displays data about each employee represented by SSN in a set or file. The output may be generated on the teletype either in a predefined format or a dynamically constructed format, or it may be produced by means of a batch run.

DISTART* Controls the display process; permits selection of a format, disposition of output, and printing of headings.

ELEMENT* Displays data about the employee whose SSN has been entered by the user at the TTY. Data will be presented as specified in DISPLAY.

FETCH Retrieves specified fields from user records, for output.

HEADER Prints headings on each page of the output.

PRINTER Prints output for specified employees in a given format.

THERE Creates a random file name for a batch job.

THE (4,0) OVERLAY (SROVL)

SROVL* The main program of the (4,0) overlay. Transfers directly to the subroutine SORT.

DEFINIT* Defines and saves a new format, for use in addition to the predefined formats, in printing output. This subroutine duplicates the DEFINE subroutine included in OVERLAY (3,0).

DISTART* Controls the display process; selects a format for printing, designates disposition of output, and determines whether or not headings are desired. This subroutine duplicates the DISTART subroutine in OVERLAY (3,0).

SORT* Coordinates the preparation of the data to be sorted by a batch program. Retrieves the category to be sorted, provides for transfer to the HELP, TIME, and ABORT subroutines, allows definition of sort type, predefined or dynamically defined — makes up the input for the job, and submits this input to the batch queue.

SORTDEF* Prepares a sort definition from one or more fields and catalogs the definition. This sort definition may be used for sorting in the same way as the predefined formats are used.

THERE Creates a random file name for a batch job.

THE (5,0) OVERLAY (CMOVL)

CMOVL* — The main program of the (5,0) overlay. It transfers control directly to the subprogram COMPUTE.

CANDER — Writes out a series of records which are pointed to by the words KBR_j. These records are modified forms of records $TBLB_i$: all words in records pointed to by KBB_j that match words in records pointed to by KBA_i are indicated. CANDER operates in the same way as ANDER of the (2,0) overlay.

CGROUP* — Computes the number of employee SSNs occurring in both a constructed set and a specified category. Computes the sum of the values of each element in the resultant set.

COMPUTE* — The controlling subroutine of this overlay. It interprets user input-commands and category names. Verifies existence of inverted category. Transfers to CGROUP and, on return, prints out the computed data.

INUM — Computes a right-adjusted integer from a left-adjusted alphanumeric character.

MATCH — Called on by CANDER to perform the same function as the like-named subroutine in the (2,0) overlay. Compares elements with a given header value in one set with all elements of another set, regardless of header value. Indicates a match in the matching words of the latter set.

THE (6,0) OVERLAY (TBOVL)

TBOVL* — The main program of the (6,0) overlay. It transfers directly to the subroutine TABLE.

COMPARE — Compares two input numbers to determine whether one is greater than, equal to, or less than the other. (COMPASS subroutine)

COUNTER — Counts the number of elements of a particular category having a specific header value, and, at the same time, some specified header value in a second category.

TABLE* — Gets category names. determines file and record numbers, and reads in key blocks; calls the subroutine TBLCNT to generate and print out the table, or, if the output is to be generated by a batch program, TABLE prepares a file of input and calls THER4.

TBLCNT — Computes and prints out a table giving the correspondence of two categories by header value. The number of elements (in each of the two categories) which possess each combination of header values is printed out.

THER4 — Creates a random file name for a batch job.

THE (7,0) OVERLAY (HLPOVL1)

HLPOVL1 — This program, called by the main overlay, determines which of the three subroutines, CATHELP, SPCHELP, and VALHELP, is to be invoked, and transfers to it. These and other HELP subroutines called by them are described in the section entitled "On-Line Information about QUERY Commands and Directives."

NUMBER — A subroutine called by SETLOOP. It reads a number entered at the terminal and converts it to binary number.

SETLOOP — This subroutine determines which of the values of a category are to be displayed by DISHELP, DEFHLEP, or VALHELP.

THE (10,0) OVERLAY (HLPOVL2)

HLPOVL2 — This program, called by the main overlay, transfers to the COMHELP subroutine which provides general information about all of the valid commands of the QUERY system. COMHELP and the subroutines it calls are described in the section "On-Line Information about QUERY Commands and Directives."

THE (11,0) OVERLAY (BTCH)

BTCH — This program calls library subroutines for help in initializing a batch run to produce output for programs DSOVL, SROVL, and TBOVL. It computes parameters which are necessary to complete the control cards for the batch run, and stores these in a common block. The program also designates an output unit as TAPE3. The following library subroutines are called:

CLUNLD — Calls the subroutine CLUXX to close and unload a file.

GETCOM — Calls the subroutine GETSET to read from the COMRADE[23] common area, SUBCOM.

LOCATE — Locates a character string of shorter or equal length within a longer string.

MOVE — Moves specific-length character strings between arrays.

ZPFMAC — Performs operations, such as ATTACH, CATALOG, PURGE, and REQUEST, on permanent files.

THE (12,0) OVERLAY (REPBTCH), THE (13,0) OVERLAY (SRTBTCH), AND THE (14,0) OVERLAY (TBLBTCH)

These programs submit batch jobs for programs DSOVL, SROVL, and TBOVL, respectively, after generating a record of control cards. The output file is cataloged and unloaded, control cards are set up, a choice is made as to the destination of the output (printer or user-terminal), and a control card record is submitted to the batch input stream. Library subroutines called by each program include BCHIN and CHOICE, described following, and CLUNLD and ZPFMAC, described on the previous page.

BCHIN — Submits a control card record to the batch queue.

CHOICE — Allows the user to choose the destination of his batch output.

ROUTINES IN THE INVERT PROGRAM

INVERT* — The main program. It performs as follows:

1) Opens a mass storage file, calls the subprogram DATADIV, and writes out the generated formats to disk.
2) Initializes the IS system.
3) Reads the 5-word formats.
4) Does steps 4-9 for each field (format) to be "inverted on." Reads parameters for a field.
5) Gets the array of formats for the current field.
6) Unpacks the fifth word of the format.
7) Makes up elements of the key block.
8) Sorts all employee records by header values of the current category (name of field).
9) Completes and writes to tape (disk) the key block. Forms the key control block for the current category.
10) Forms and writes to disk a control record for the total inversion.
11) For each key, prints key control block, key block, header blocks, and data blocks.
12) Terminates the inverter.

BINSORT — Arranges (sorts) the 5-word formats constructed by DATADIV into alphabetical order by name (the first two elements of the format).

BINSRCH — Determines whether a particular input name (or a format) is in a given array. Specifically, in INVERT, BINSRCH determines whether or not a particular category name (COBNAME) exists in the array of names generated by DATADIV.

COMPARE — Compares two inputs to determine a greater than, equal, or less than relationship.

DATADIV* — Prepares a data array for subsequent processing by the INVERTER. The output of a COBOL program (data map) is read into DATADIV and a 5-word format for each of the fields occurring in the data map is made up. The name of the format is given in the first two words of the format.

FETCH — Selects a specific field from a user record (USERREC) of data.

GETDATA — Called by SORTER, this subroutine fetches from an employee record a specific field of data, as represented by a format made up in DATADIV.

JULDATE — Gets the current (Julian) date.

MERGE — Merges formats into the alphabetical order determined by COMPARE.

PARTIAL — Positions the characters of a field at the beginning of a word.

PUTDATA — Writes out the social security numbers sorted by SORTER. When the array is full, it is written as a data record to mass storage.

SORTER* — Arranges employee SSN's according to header values within a given category. Thus, if AGE is the sort category, the SSN's of all employees of age 16 are followed by those of all employees of age 17, etc. This arrangement greatly facilitates the retrieval of employee data (SSN-keyed) for an employee specified by some specific attribute (e.g., age).

STRIP — For a data word fills character positions following a blank with blanks.

TIMCALC — Calculates the time span (in years) between an input date (Julian) and the present date (Julian).

THE REPORT PROGRAM

REPORT* — Main program. Reads input data generated by QUERY by means of ordinary FORTRAN reads from TAPE1 (a disk file). Initializes IS and, for each SSN in the input, reads employee records (from file EMPDATA), and calls subroutine PRINTER.

PRINTER — Writes specified data for a designated employee to the output file.

HEADER — Writes headings on each page of the printed output.

FETCH — Retrieves from a user record the specified fields to be written out.

THE SORTREP PROGRAM

SORTREP* — The main program. It performs as follows:

1) Initializes the IS system.
2) Reads the sort and display definitions from TAPE1 using ordinary FORTRAN read statements.
3) Calls the subroutine SORTER to perform the sort and to display the results.

FETCH — Sets up a new array containing only certain specified data of the first array.

GET* — Gets information from the user record of each employee (TAG or SSN), which information is determined by the sort definition. The data items are packed in an array (IN), the first word being the SSN.

HEADER — Prints out a heading at the top of each page of output of identifying information about each field in the format.

PRINTER — Prints out data about an employee in a predetermined format.

PUT* — Controls the output of employee data as specified by the display format.

SORTER — Sorts a selected set of employees according to certain "keys." This subroutine makes use of the system subprogram SMCON7, to do the actual sorting. SORTER calls GET to fetch the data, and PUT to print it.

THE TABLE PROGRAM

TABLE* — The main program. It performs as follows:

1) Opens mass storage file.
2) Reads header and data (SSN) blocks from TAPE2, using ordinary FORTRAN read statements.

3) Writes this information to a mass storage file (TAPE1).

4) Reads the key blocks and category names from TAPE2.

5) Calls the subroutine TBLCNT to format and print out the requested table. IS is not used in program TABLE, because only inverted categories are referenced.

COMPARE — Compares two input numbers to determine whether one is greater than, equal to, or less than the other (COMPASS subroutine).

COUNTER — Counts the number of elements in one category having a specific header value and also some specific header value in a second category.

TBLCNT — Computes and prints out a table giving the correspondence of two categories by header value. The numbers of elements (in each of the two categories) which possess each combination of header values is printed out.

APPENDIX D
SPECIALLY USED CODING SYSTEMS

THE CDC INDEXED SEQUENTIAL(IS) DATA STORAGE AND RETRIEVAL SYSTEM

The indexed sequential (IS) file organization system is one of several file organization systems available under the SCOPE 3.4 operating system through the CDC Record Manager. IS files offer certain advantages of both sequential files and direct access files in that they can be processed either sequentially or randomly. The QUERY employee data file EMPDATA contains a record for each employee. The QUERY system enables the user to access information about an individual employee directly from within the subroutines DISPLAY and ELEMENT. It enables the entire EMPDATA file, or a generated subset of the file, to be accessed sequentially from within the programs INVERT and SORTREP. This capability of processing IS files both randomly and sequentially allows the user to organize his data into a more flexible and convenient file structure, although he requires more storage space and the program execution time may be increased.

Each record of an indexed sequential file is associated with an identifying "key" which is used by Record Manager in arranging the records at the time the file is created. Record Manager maintains an index of all records in the file by creating an index entry to link the record key with the file location of that record when the record is first written to the file. When a user wishes to access a record, he supplies the appropriate record key, and the desired record is returned to him in his own dimensioned working storage area. The manipulation of the index is done automatically. Since the key is used to locate an index entry, not the record itself, the key need not be part of the record.

To summarize, an indexed sequential file is a collection of index blocks of key entries, one key entry composed of a record key and a pointer to the record's location within the data block for each record contained in the block. The index block, on the other hand, contains information which links the record key with the data block containing that record. The index-block information is used for indirect access. Thus an index entry in an index block points to the data block containing the record, and a key entry within the data block points to the precise location of the record.

The following subroutines are available to the user of the IS data storage and retrieval system:

FILEIS - Uses data supplied in a list following the call to set up fields in the file information table (FIT). These data give information about structure, status, and position of the file.

OPENM - Opens a file and prepares it for further processing. Parameters in the list specify whether the file is to be read in or written out, or both.

GET - Returns a single record to the working storage area. The key to the subject record is provided by the user.

GETN - Reads the IS file as if it were a sequential file, i.e., it reads the record from the current position of the file.

THE CDC SORT/MERGE SYSTEM

The use of the SORT/MERGE program SMCON7 under the SCOPE 3.4 operating system is considerably different than under SCOPE 3.3. It has been designed for use with a series of macro calls which are expanded by the COMPASS assembler. SMCON7 is called by COMPASS subroutines of the programs INVERT and SORTREP which set up necessary parameters for the call and contain coding necessary for calling data acquisition and disposition routines.

The SORT/MERGE macros called by each subroutine are SORT, KEY and OWNCODE. The SORT macro initiates SORT/MERGE functions. The KEY macro specifies each sort key used during the sort process (a "key" being a field of data used for sorting data records according to a specified order). A separate KEY macro call must be made for each sort key specified, and as many as 100 sort keys are permitted in any one run. The keys are processed in the order given. Parameters included with this macro specify the first byte and bit of the sort key, the number of complete bytes, any extra bits in the key, and, if needed, the type of data to be processed, the name of a user-specified collating sequence, and the sequencing order of sort processing (ascending or descending).

The OWNCODE macro specifies entry point names of a user's own code exits, each of which calls in another subroutine to perform a specific function, such as the retrieval or printing of records, or the termination of the sort. In PROGRAM INVERT user-coded subroutine GETDATA is referenced by the OWNCODE macro to retrieve data that are used by SMCON7. Subroutine PUTDATA stores sorted records into a user area, from which they eventually will be picked up for printing.

FORTRAN MASS STORAGE SUBROUTINES

When large quantities of data are to be stored on disk, object-time subroutines are used to control the transmission of records to and from central memory. A mass storage file of the QUERY system is referenced by the following statements:

```
CALL OPENMS (u, ix, l, p)
CALL READMS (u, fwa, n, i)
CALL WRITMS (u, fwa, n, i)
```

where

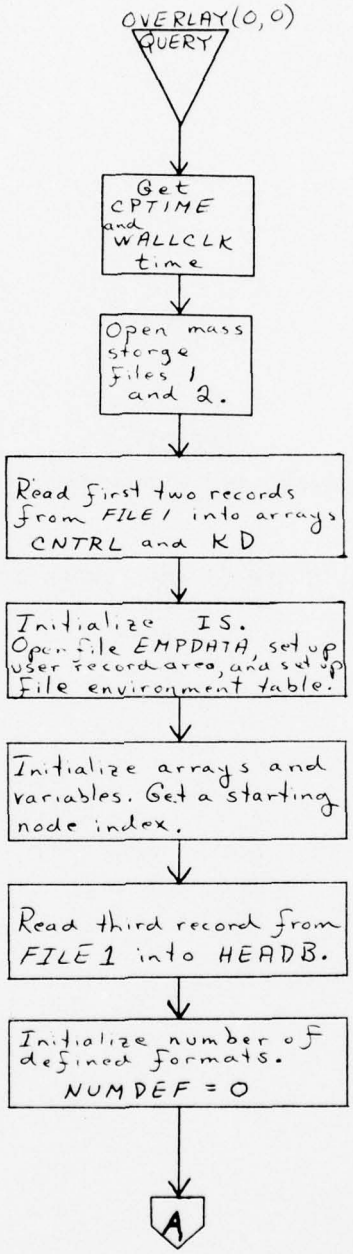
u	logical unit number.
ix	address of the first word of the record index in central memory.
l	length of the index array.
p	indicates that the file is referenced through a name index (p=1) or a number index (p=0).
fwa	central memory address of the first word of the record.
n	number of central memory words to be transferred.
i	record number or address of a call containing the record name.

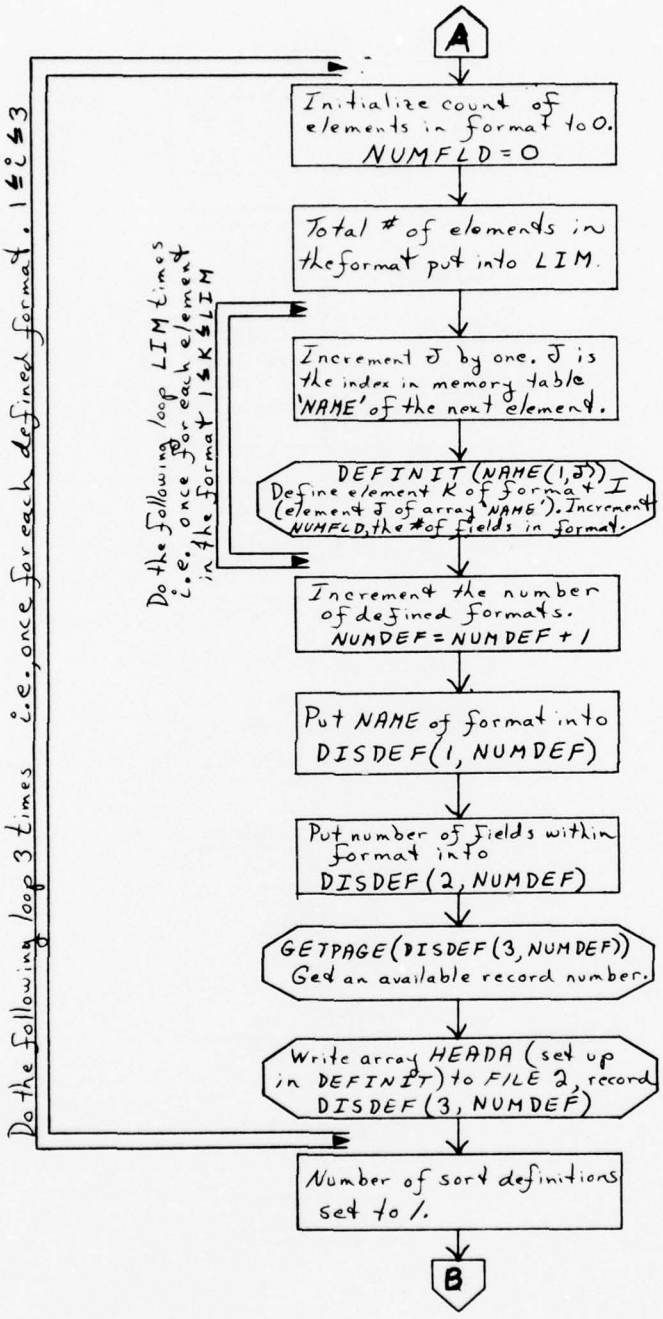
Before processing can proceed, the mass storage file must be opened (OPENMS). The operating system, SCOPE, operates upon the file in a random access mode. Subroutines READMS and WRITMS perform the actual transfer of data to and from central memory. After the job terminates, the mass storage file is automatically closed.

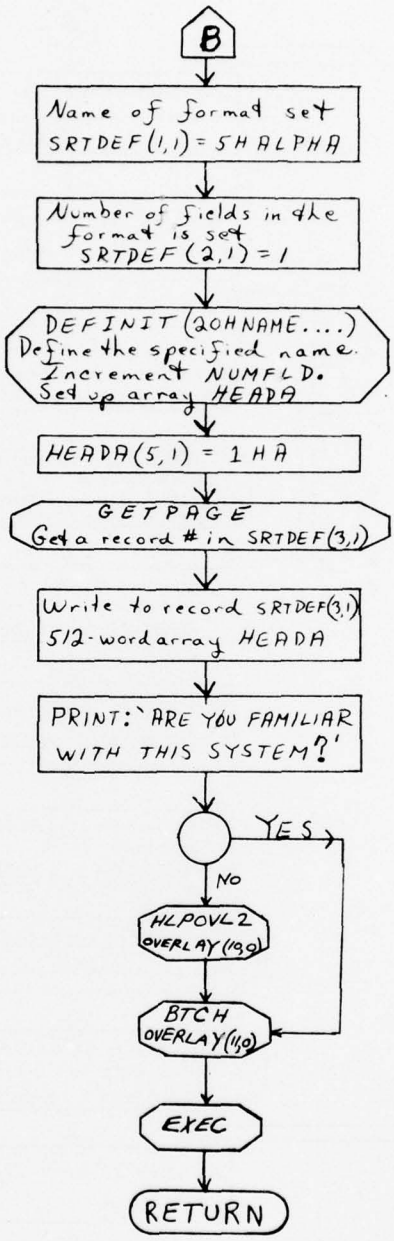
APPENDIX E

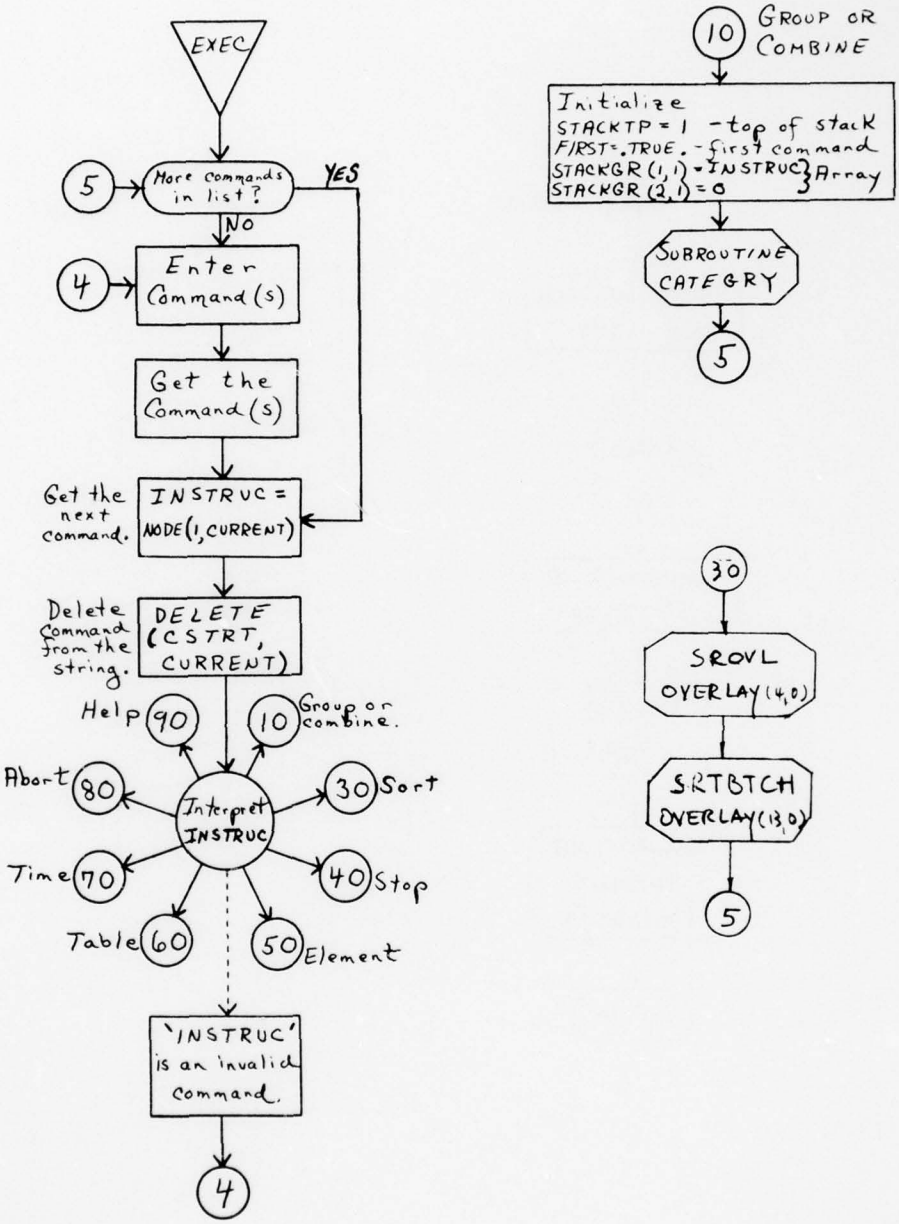
FLOW CHARTS OF PROGRAMS AND SUBROUTINES IN THE QUERY SYSTEM

The following flowcharts are of the larger programs and subroutines of the QUERY system.


OVERLAY(0,0)
QUERY
Get CPTIME and WALLCLK time
Open mass storge files 1 and 2.
Read first two records from FILE1 into arrays CNTRL and KD
Initialize IS. Open file EMPDATA, set up user record area, and set up File environment table.
Initialize arrays and variables. Get a starting node index.
Read third record from FILE1 into HEADB.
Initialize number of defined formats. NUMDEF = 0
A


A
Initialize count of elements in format to 0.
NUMFLD = 0
Total # of elements in the format put into LIM.
Increment J by one. J is the index in memory table 'NAME' of the next element.
DEFINIT (NAME(1,J))
Define element K of format I (element J of array 'NAME'). Increment NUMFLD, the # of fields in format.
Do the following loop LIM times i.e. once for each element in the format 1≤K≤LIM
Increment the number of defined formats.
NUMDEF = NUMDEF + 1
Put NAME of format into DISDEF(1, NUMDEF)
Put number of fields within format into DISDEF(2, NUMDEF)
GETPAGE(DISDEF(3, NUMDEF))
Get an available record number.
Write array HEADA (set up in DEFINIT) to FILE 2, record DISDEF(3, NUMDEF)
Number of sort definitions set to 1.
B
Do the following loop 3 times i.e. once for each defined format. 1≤I≤3

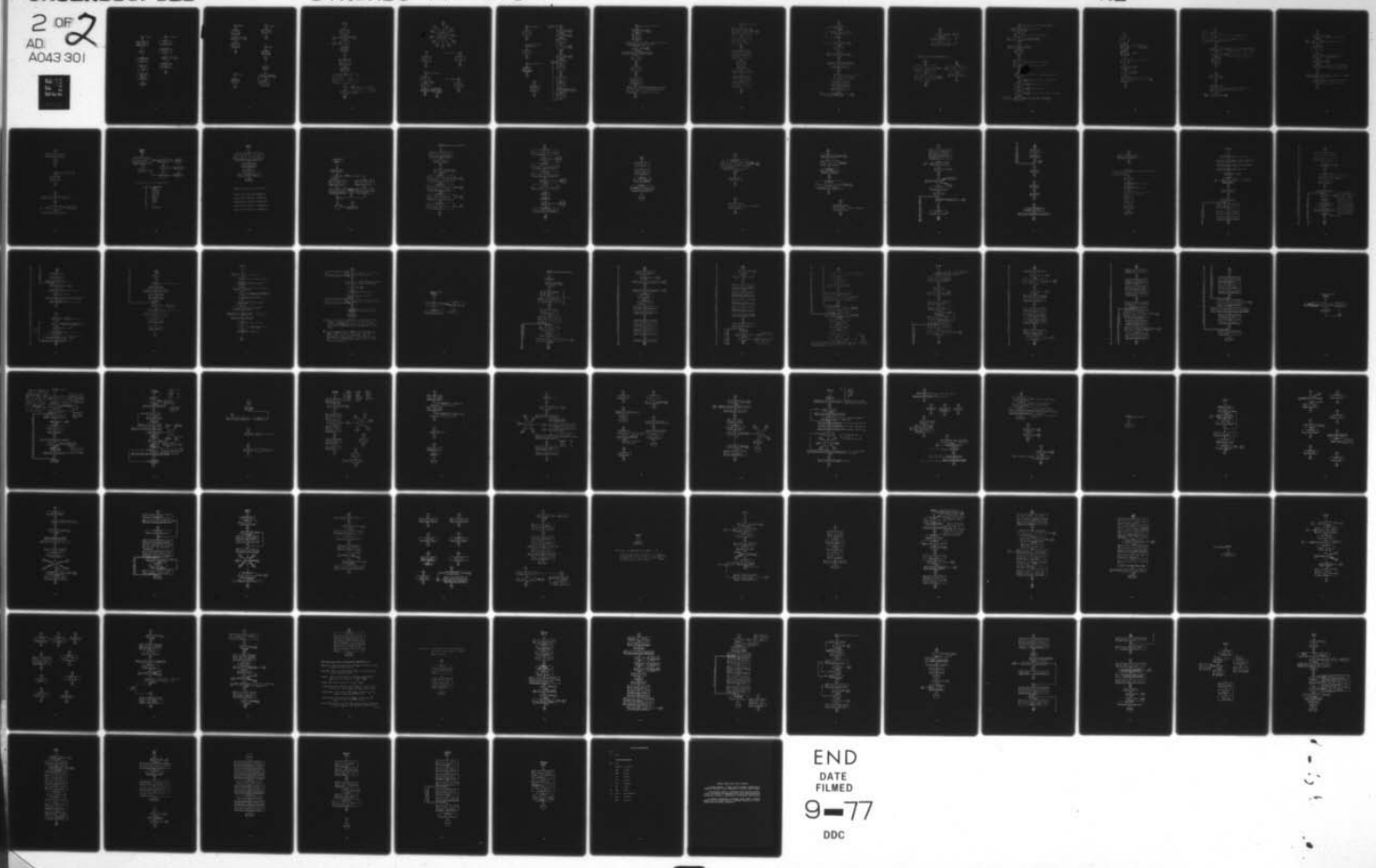

B
Name of format set
SRTDEF(1,1) = 5HALPHA
Number of fields in the format is set
SRTDEF(2,1) = 1
DEFINIT (20HNAME....)
Define the specified name.
Increment NUMFLD.
Set up array HEADA
HEADA(5,1) = 1HA
GETPAGE
Get a record # in SRTDEF(3,1)
Write to record SRTDEF(3,1)
512-word array HEADA
PRINT: `ARE YOU FAMILIAR WITH THIS SYSTEM?'
YES
NO
HLPOVL2
OVERLAY(10,0)
BTCH
OVERLAY(11,0)
EXEC
RETURN


EXEC
5
More commands in list?
YES
NO
4
Enter Command(s)
Get the Command(s)
Get the next command.
INSTRUC = MODE(1,CURRENT)
Delete command from the string.
DELETE (CSTRT, CURRENT)
Help 90
10 Group or combine.
Abort 80
Interpret INSTRUC
30 Sort
Time 70
40 Stop
Table 60
50 Element
'INSTRUC' is an invalid command.
4
10 GROUP OR COMBINE
Initialize
STACKTP = 1 - top of stack
FIRST=.TRUE. - first command
STACKGR(1,1) = INSTRUC } Array
STACKGR(2,1) = 0
SUBROUTINE CATEGRY
5
30
SROVL
OVERLAY(4,0)
SRTBTCH
OVERLAY(13,0)
5

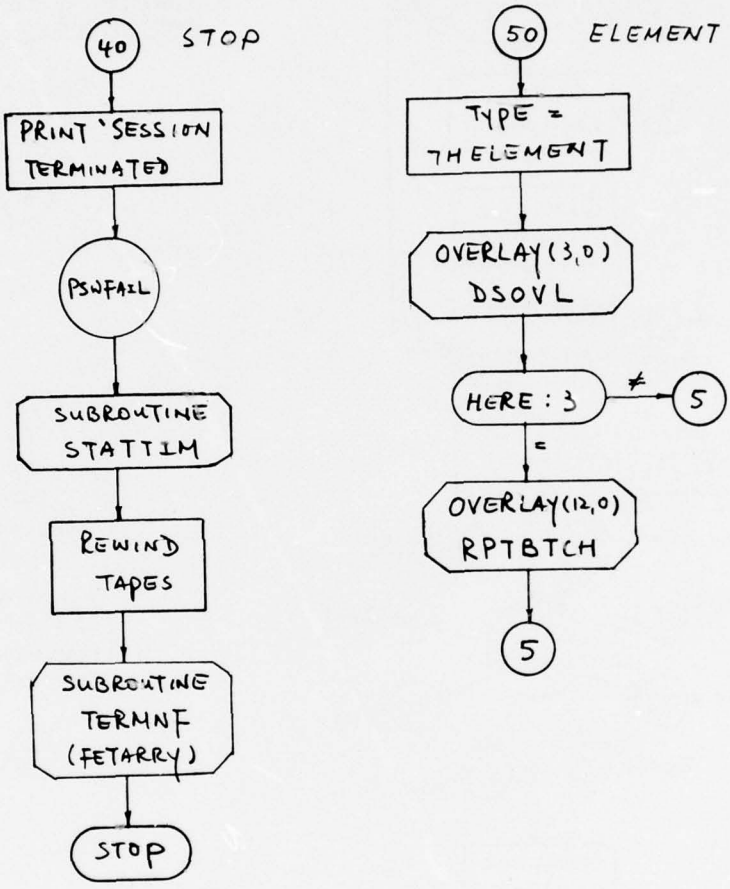
AD-A043 301 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 5/8
THE QUERY SYSTEM - A COMPONENT OF THE DTNSRDC PERSONNEL DATA MA--ETC(U)
JAN 77 P E BATTEY

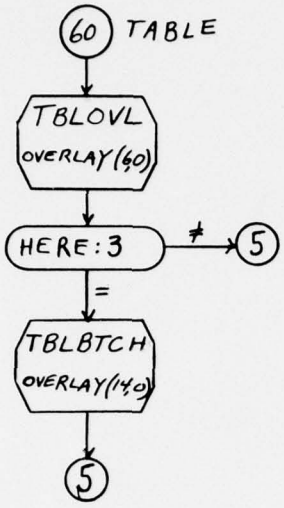
UNCLASSIFIED DTNSRDC-77-0078 NL

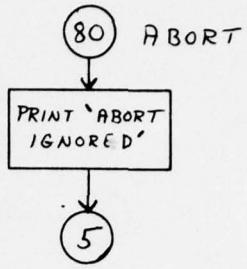
2 OF 2
AD
A043 301

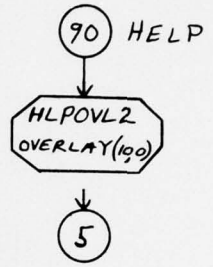


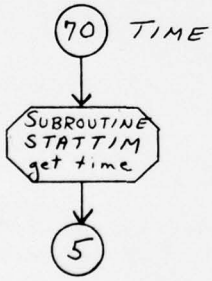
END
DATE FILMED
9-77
DDC

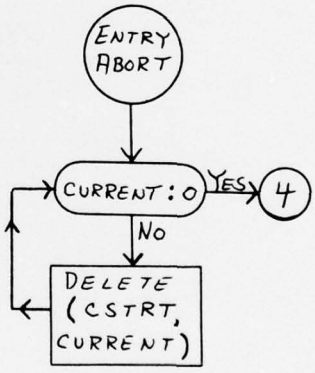

40
STOP
PRINT 'SESSION TERMINATED
PSWFAIL
SUBROUTINE STATTIM
REWIND TAPES
SUBROUTINE TERMNF (FETARRY)
STOP
50
ELEMENT
TYPE = THELEMENT
OVERLAY(3,0) DSOVL
HERE : 3
≠
5
=
OVERLAY(12,0) RPTBTCH
5

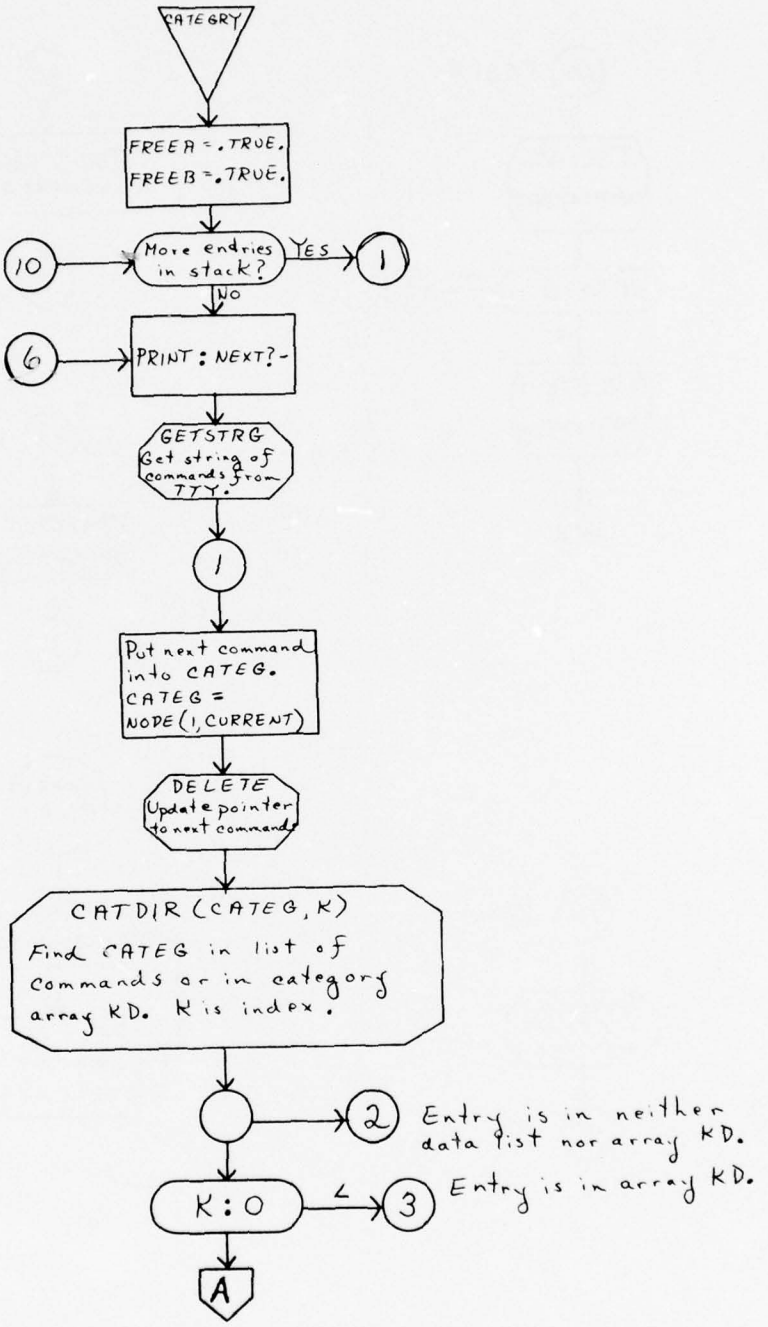

60
TABLE
TBLOVL
OVERLAY(60)
HERE:3
≠
5
=
TBLBTCH
OVERLAY(140)
5

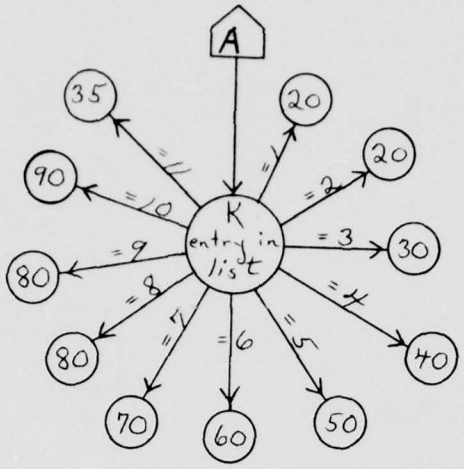

80
ABORT
PRINT 'ABORT IGNORED'
5

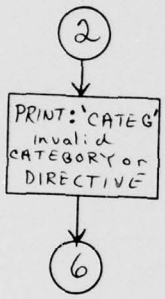

90
HELP
HLPOVL2
OVERLAY(100)
5


70
TIME
SUBROUTINE STATTIM
get time
5

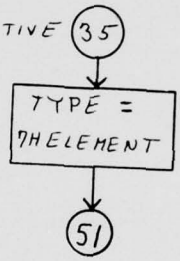

ENTRY
ABORT
CURRENT:0
YES
4
NO
DELETE
(CSTRT,
CURRENT)

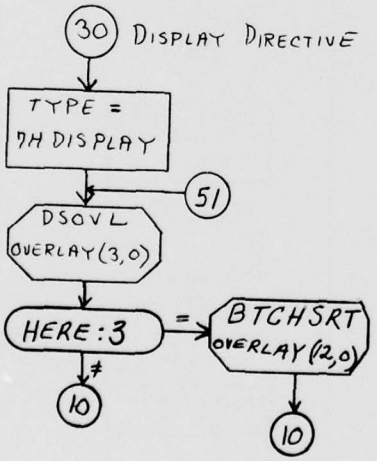

CATEGRY
FREEA = .TRUE.
FREEB = .TRUE.
10
More entries in stack?
YES
1
NO
6
PRINT : NEXT? -
GETSTRG
Get string of commands from TTY.
1
Put next command into CATEG.
CATEG = NODE(1, CURRENT)
DELETE
Update pointer to next command
CATDIR (CATEG, K)
Find CATEG in list of commands or in category array KD. K is index.
2
Entry is in neither data list nor array KD.
K : 0
3
Entry is in array KD.
A



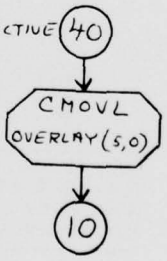


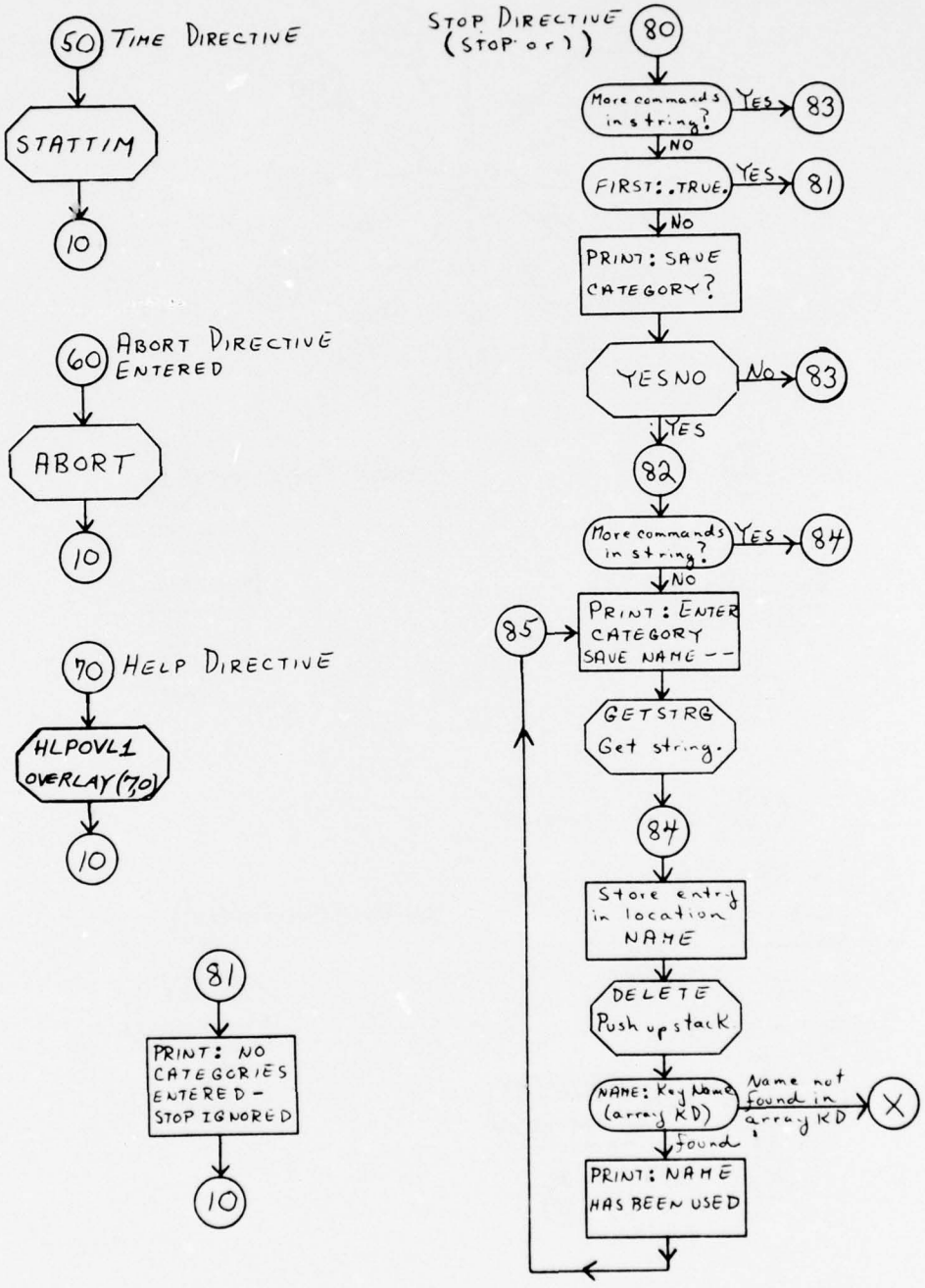
ELEMENT DIRECTIVE

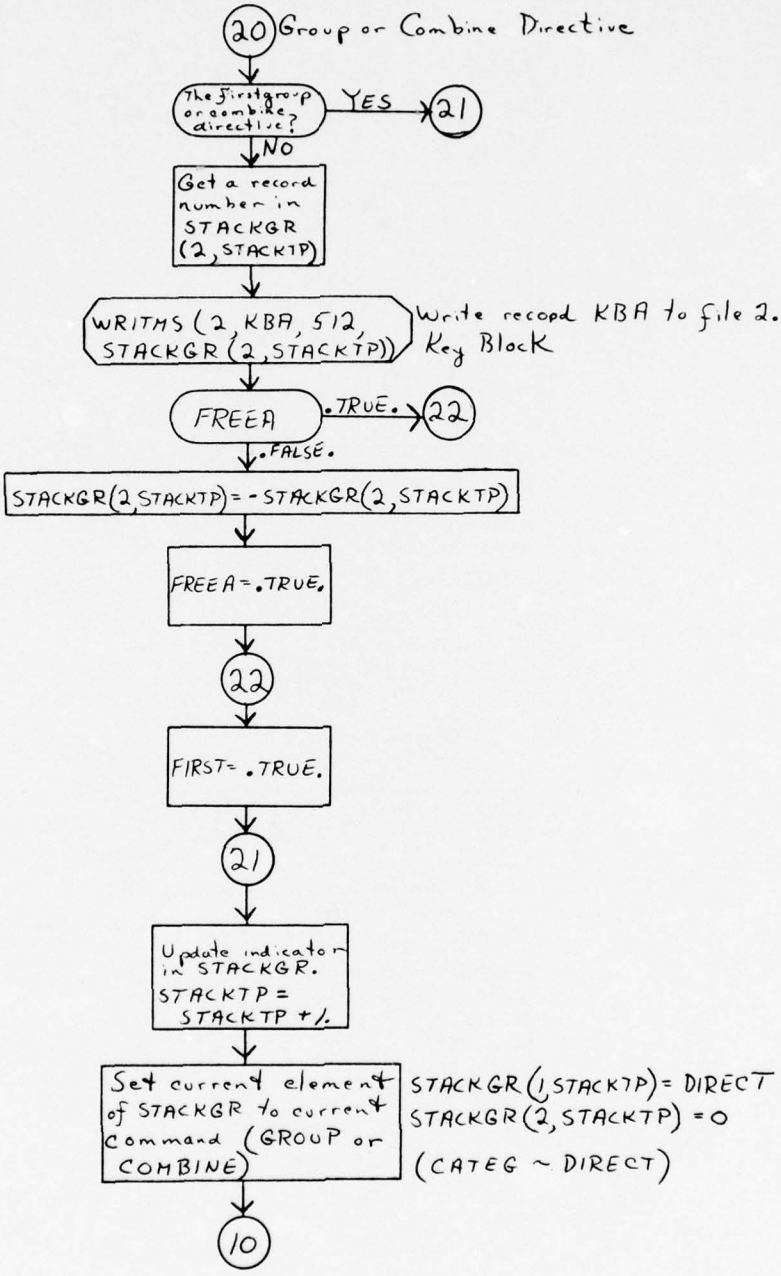


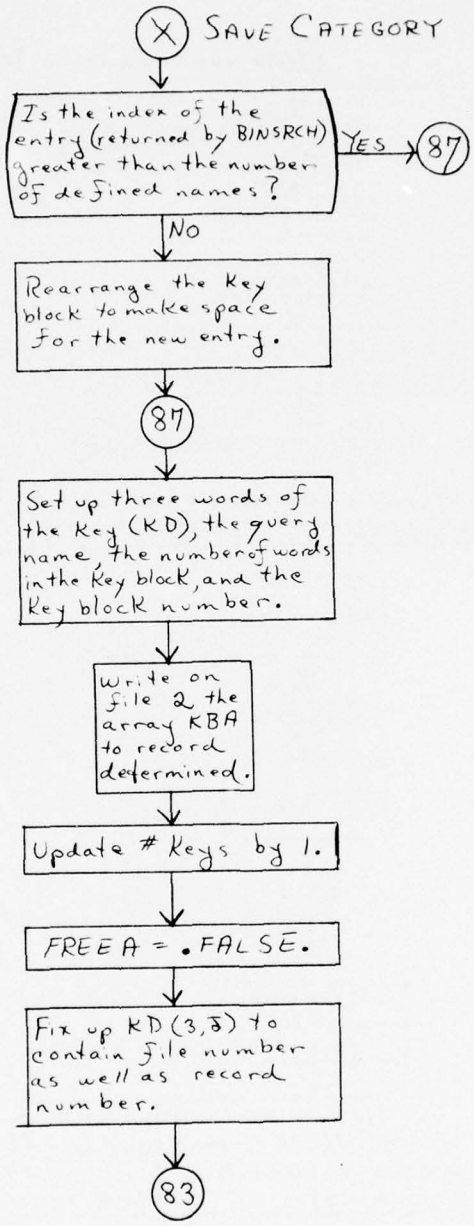


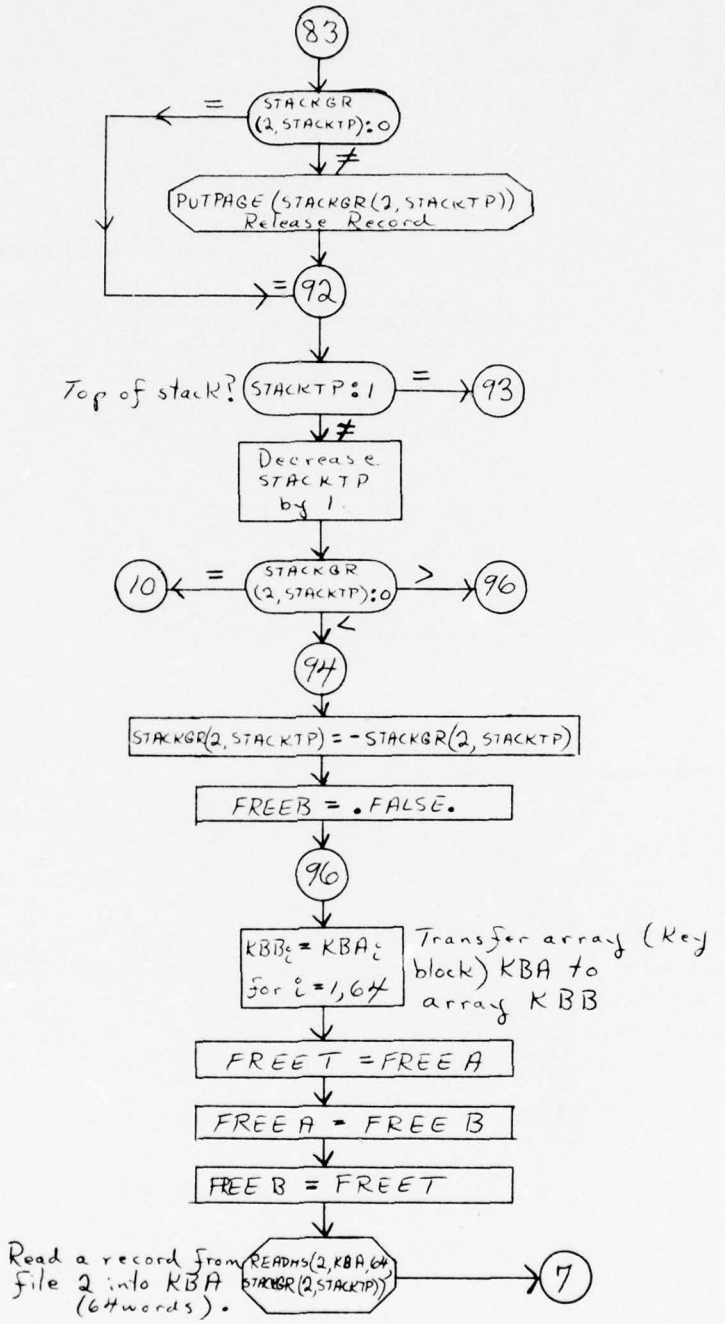
COMPUTE DIRECTIVE

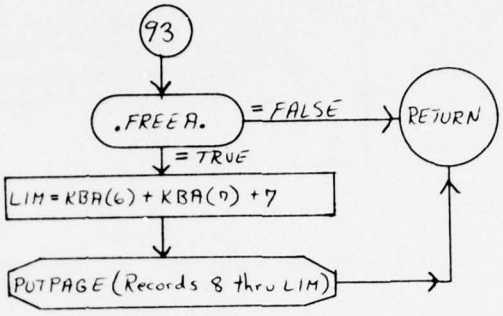



50
TIME DIRECTIVE
STATTIM
10
60
ABORT DIRECTIVE ENTERED
ABORT
10
70
HELP DIRECTIVE
HLPOVL1
OVERLAY(7,0)
10
81
PRINT: NO CATEGORIES ENTERED - STOP IGNORED
10
STOP DIRECTIVE (STOP or))
80
More commands in string?
YES
83
NO
FIRST: .TRUE.
YES
81
NO
PRINT: SAVE CATEGORY?
YESNO
NO
83
YES
82
More commands in string?
YES
84
NO
85
PRINT: ENTER CATEGORY SAVE NAME --
GETSTRG
Get string.
84
Store entry in location NAME
DELETE
Push up stack
NAME: Key Name (array KD)
Name not found in array KD
X
found
PRINT: NAME HAS BEEN USED

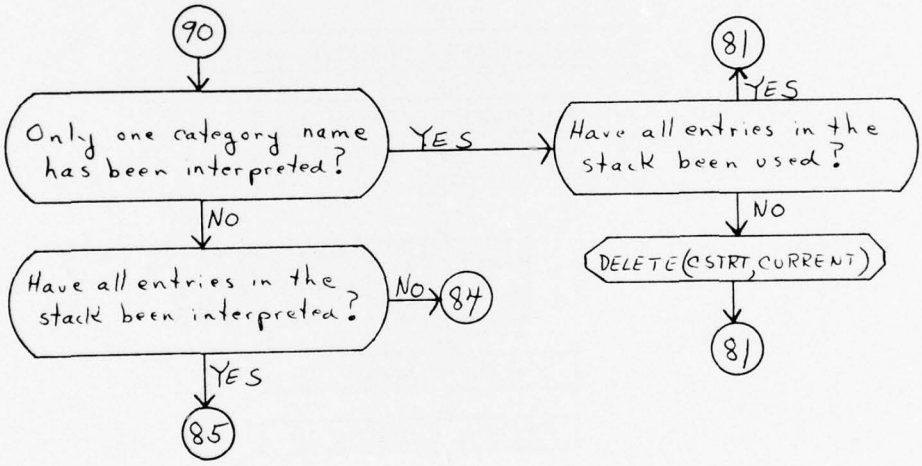

20
Group or Combine Directive
The first group or combine directive?
YES
21
NO
Get a record number in STACKGR (2, STACKTP)
WRITMS (2, KBA, 512, STACKGR (2, STACKTP))
Write record KBA to file 2. Key Block
FREEA
.TRUE.
22
.FALSE.
STACKGR(2,STACKTP) = -STACKGR(2,STACKTP)
FREEA = .TRUE.
22
FIRST = .TRUE.
21
Update indicator in STACKGR. STACKTP = STACKTP +/-
Set current element of STACKGR to current command (GROUP or COMBINE)
STACKGR(1,STACKTP) = DIRECT
STACKGR(2,STACKTP) = 0
(CATEG ~ DIRECT)
10

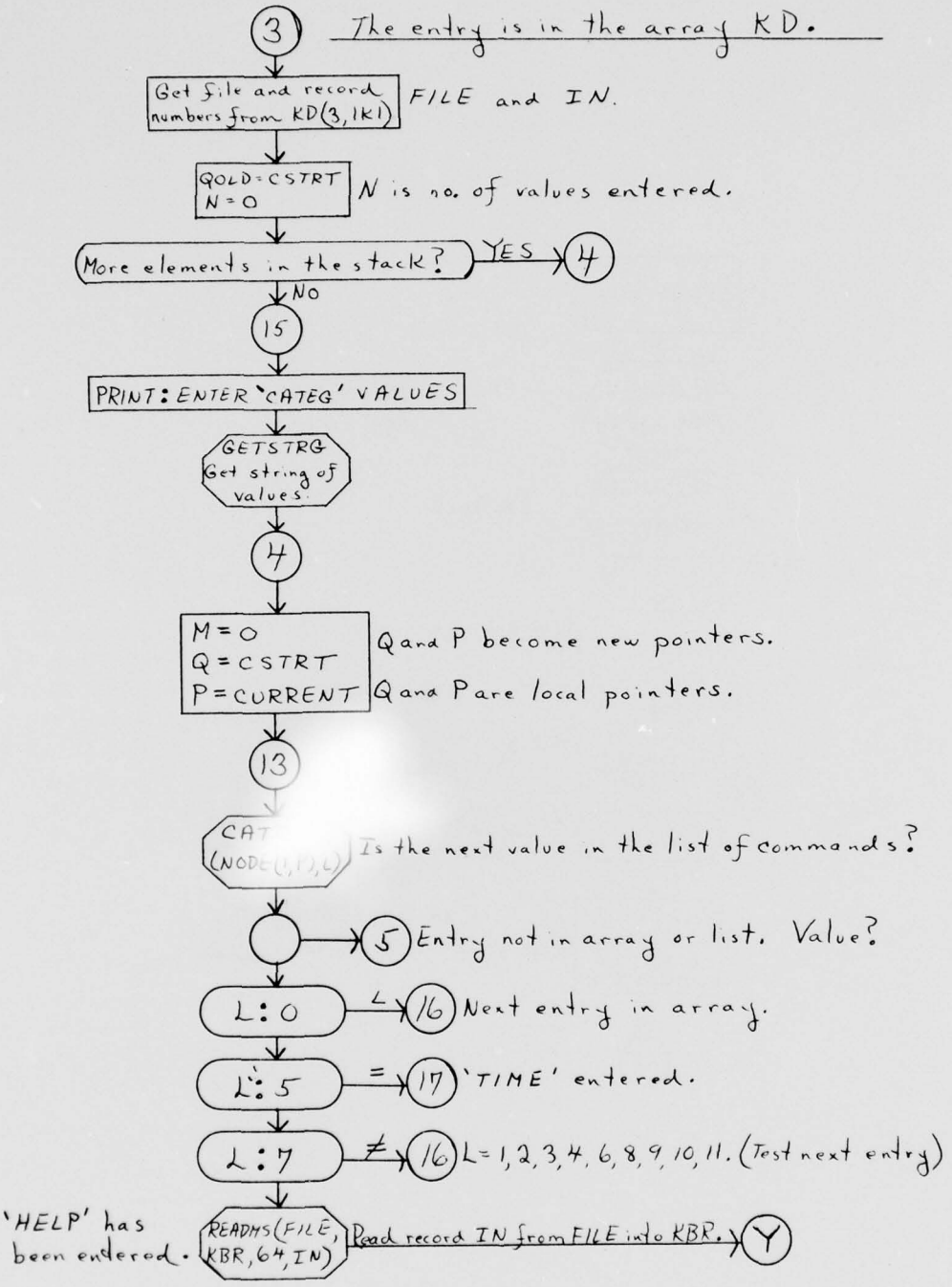

X
SAVE CATEGORY
Is the index of the entry (returned by BINSRCH) greater than the number of defined names?
YES
87
NO
Rearrange the key block to make space for the new entry.
87
Set up three words of the key (KD), the query name, the number of words in the key block, and the key block number.
Write on file 2 the array KBA to record determined.
Update # Keys by 1.
FREEA = .FALSE.
Fix up KD(3,3) to contain file number as well as record number.
83

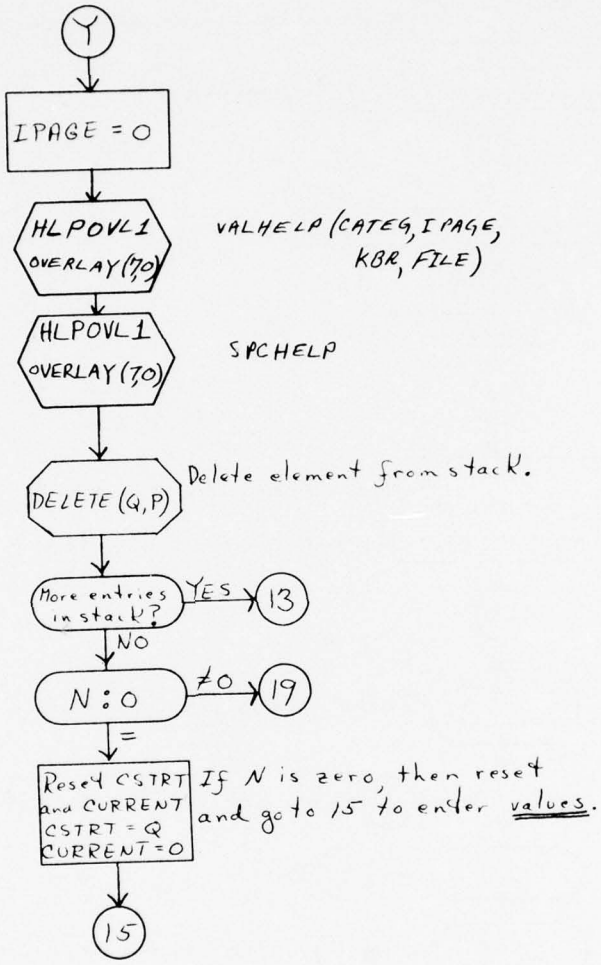

83
STACKGR (2, STACKTP): 0
=
≠
PUTPAGE (STACKGR(2, STACKTP)) Release Record
=92
Top of stack?
STACKTP: 1
=
93
≠
Decrease STACKTP by 1.
10
=
STACKGR (2, STACKTP): 0
>
96
<
94
STACKGR(2, STACKTP) = -STACKGR(2, STACKTP)
FREEB = .FALSE.
96
KBB_i = KBA_i for i = 1, 64
Transfer array (Key block) KBA to array KBB
FREET = FREEA
FREEA = FREEB
FREEB = FREET
Read a record from File 2 into KBA (64 words).
READMS(2, KBA, 64, STACKGR(2, STACKTP))
7

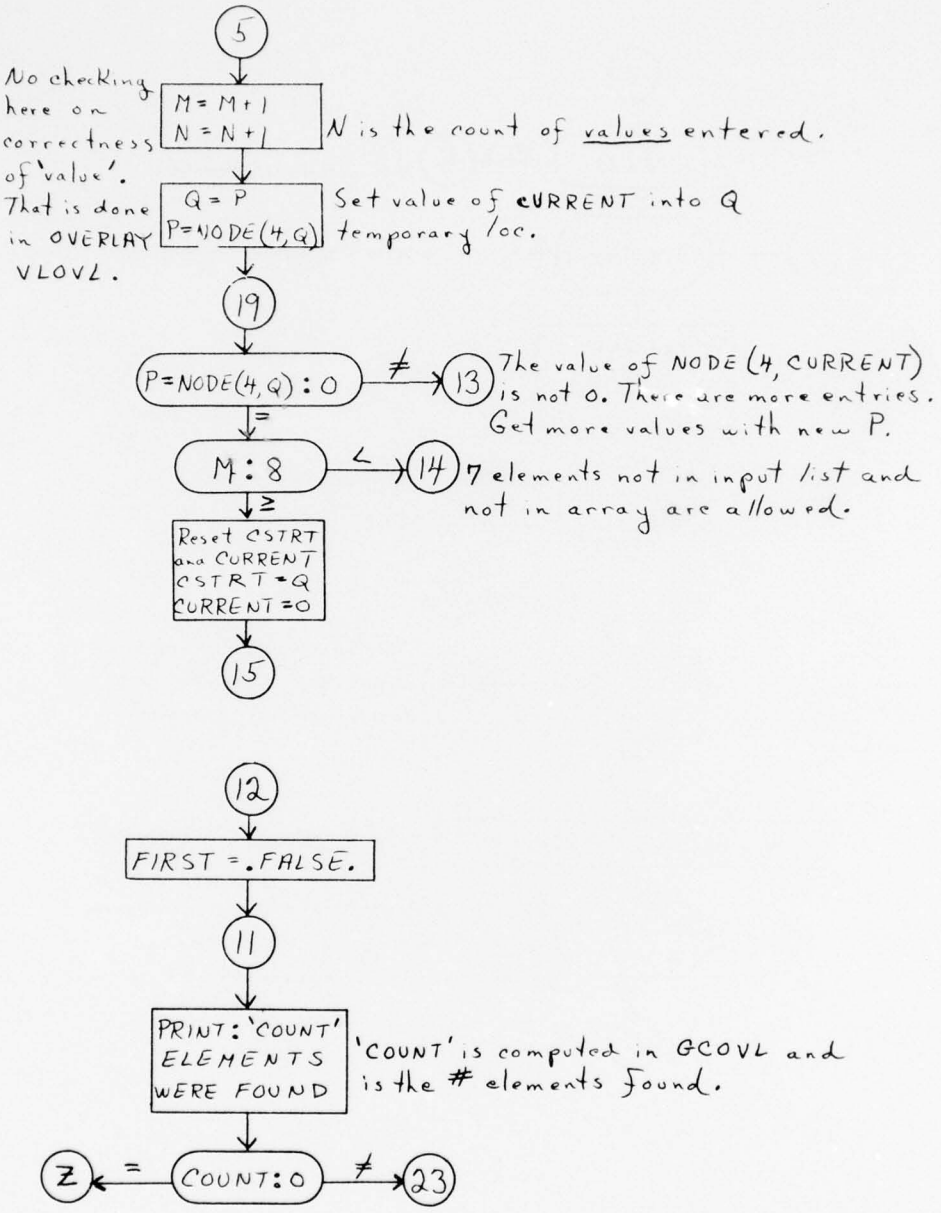


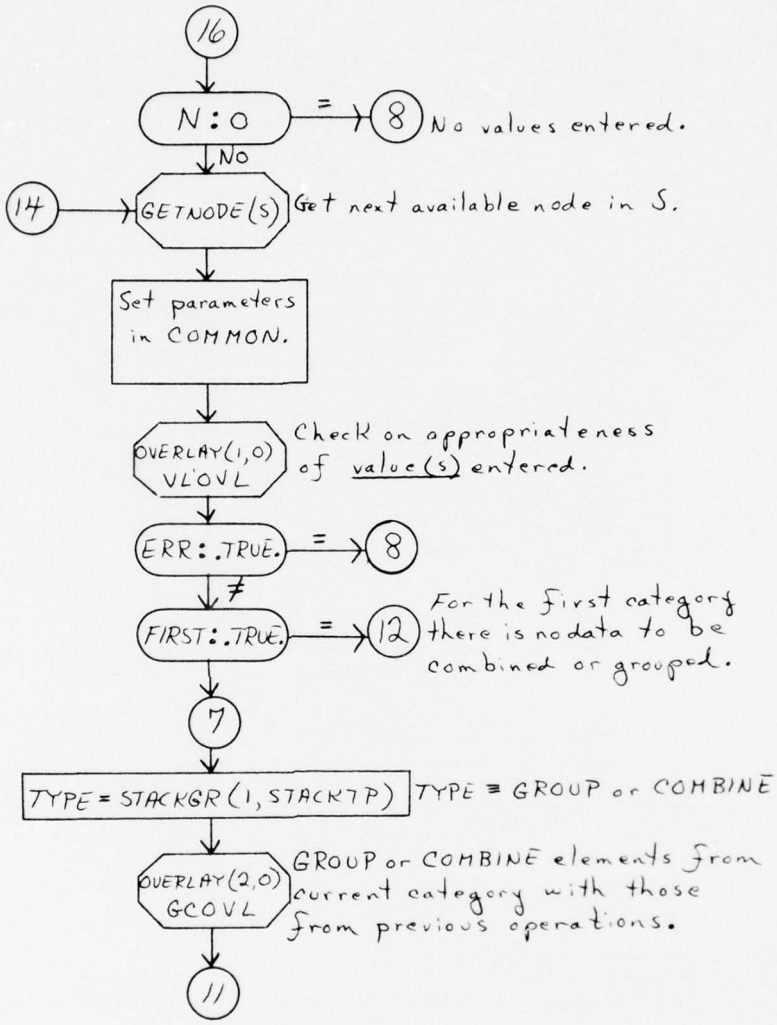
A SAVE IS TO BE PERFORMED (=)

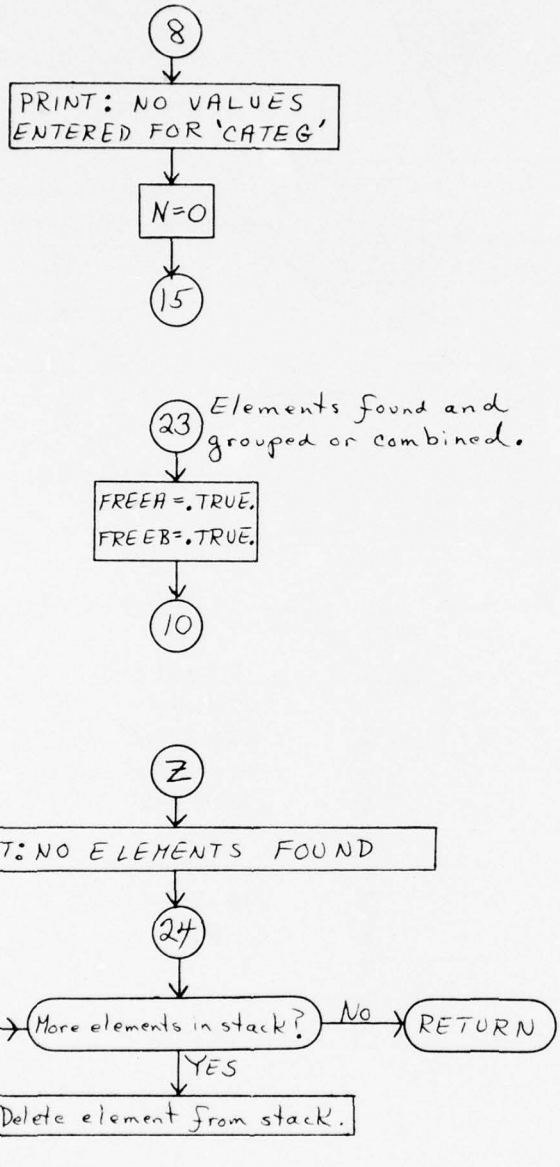


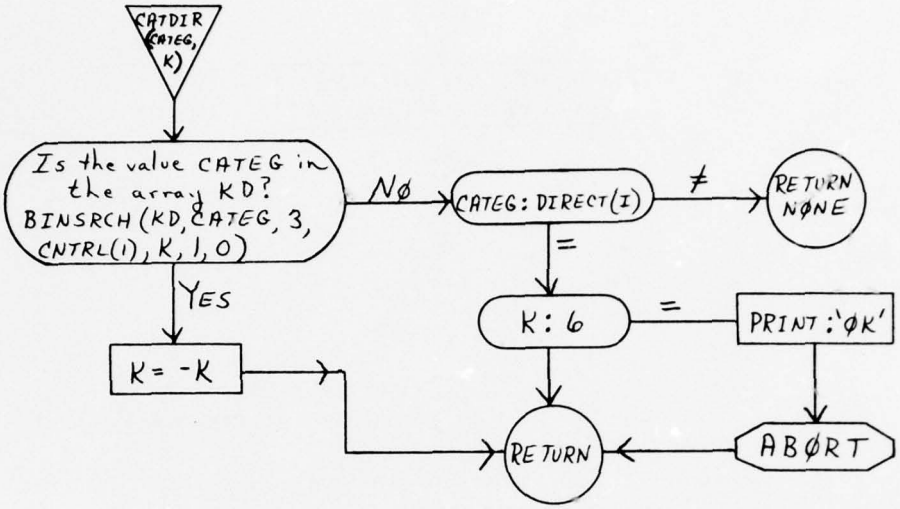

3
The entry is in the array KD.
Get file and record numbers from KD(3,1K1)
FILE and IN.
QOLD=CSTRT
N=0
N is no. of values entered.
More elements in the stack?
YES
4
No
15
PRINT: ENTER 'CATEG' VALUES
GETSTRG
Get string of values.
4
M=0
Q=CSTRT
P=CURRENT
Q and P become new pointers.
Q and P are local pointers.
13
CAT
(NODE(1,T),L)
Is the next value in the list of commands?
5
Entry not in array or list. Value?
L:0
<
16
Next entry in array.
L:5
=
17
'TIME' entered.
L:7
≠
16
L=1,2,3,4,6,8,9,10,11. (Test next entry)
'HELP' has been entered.
READMS(FILE, KBR,64,IN)
Read record IN from FILE into KBR.
Y


Y
IPAGE = 0
HLPOVL1
OVERLAY(7,0)
VALHELP (CATEG, IPAGE, KBR, FILE)
HLPOVL1
OVERLAY(7,0)
SPCHELP
DELETE (Q,P)
Delete element from stack.
More entries in stack?
YES
13
NO
N : 0
≠0
19
=
Reset CSTRT and CURRENT
CSTRT = Q
CURRENT = 0
If N is zero, then reset and go to 15 to enter values.
15


5
No checking here on correctness of 'value'. That is done in OVERLAY VLOVL.
M = M + 1
N = N + 1
N is the count of values entered.
Q = P
P = NODE(4, Q)
Set value of CURRENT into Q temporary loc.
19
P = NODE(4, Q) : 0
≠
13
The value of NODE(4, CURRENT) is not 0. There are more entries. Get more values with new P.
=
M : 8
<
14
7 elements not in input list and not in array are allowed.
≥
Reset CSTRT and CURRENT
CSTRT = Q
CURRENT = 0
15
12
FIRST = .FALSE.
11
PRINT: 'COUNT' ELEMENTS WERE FOUND
'COUNT' is computed in GCOVL and is the # elements found.
Z
=
COUNT : 0
≠
23

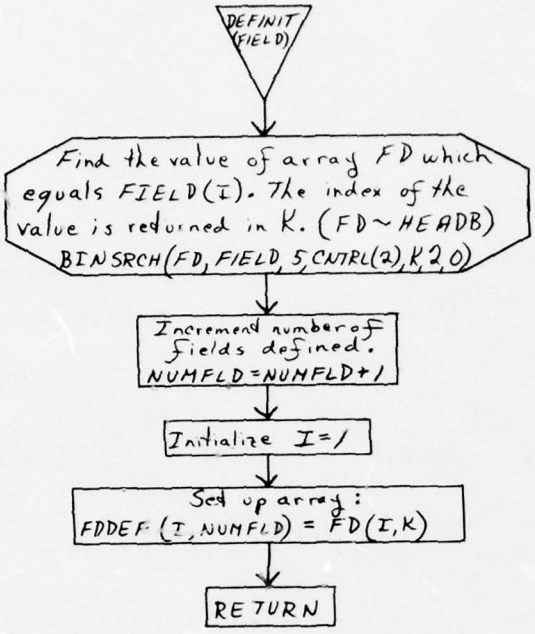

16
N:0
=
8
No values entered.
No
14
GETNODE(S)
Get next available node in S.
Set parameters in COMMON.
OVERLAY(1,0) VLOVL
Check on appropriateness of value(s) entered.
ERR:.TRUE.
=
8
≠
FIRST:.TRUE.
=
12
For the first category there is no data to be combined or grouped.
7
TYPE = STACKGR(1, STACKTP)
TYPE = GROUP or COMBINE
OVERLAY(2,0) GCOVL
GROUP or COMBINE elements from current category with those from previous operations.
11


8
PRINT: NO VALUES ENTERED FOR 'CATEG'
N=0
15
23
Elements found and grouped or combined.
FREEA=.TRUE.
FREEB=.TRUE.
10
Z
PRINT: NO ELEMENTS FOUND
24
More elements in stack?
No
RETURN
YES
Delete element from stack.



Permissible values of 'DIRECT(I)':

I	DIRECT(I)
1	GRØUP
2	CØMBINE
3	DISPLAY
4	CØMPUTE
5	TIME
6	ABØRT
7	HELP
8	STØP
9	)
10	=
11	ELEMENT



HEADA (I, J) ~ DEF(I) ~ FDDEF (I, J)

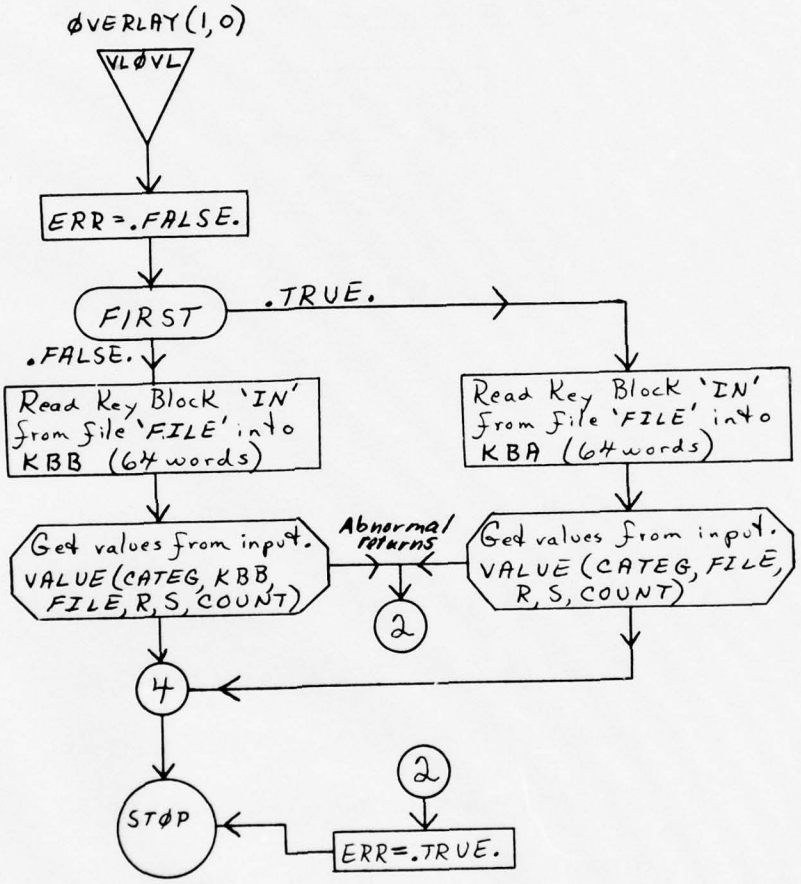
FDDEF (1, 1) ← FD(1, K) ~ HEADB (1, K)

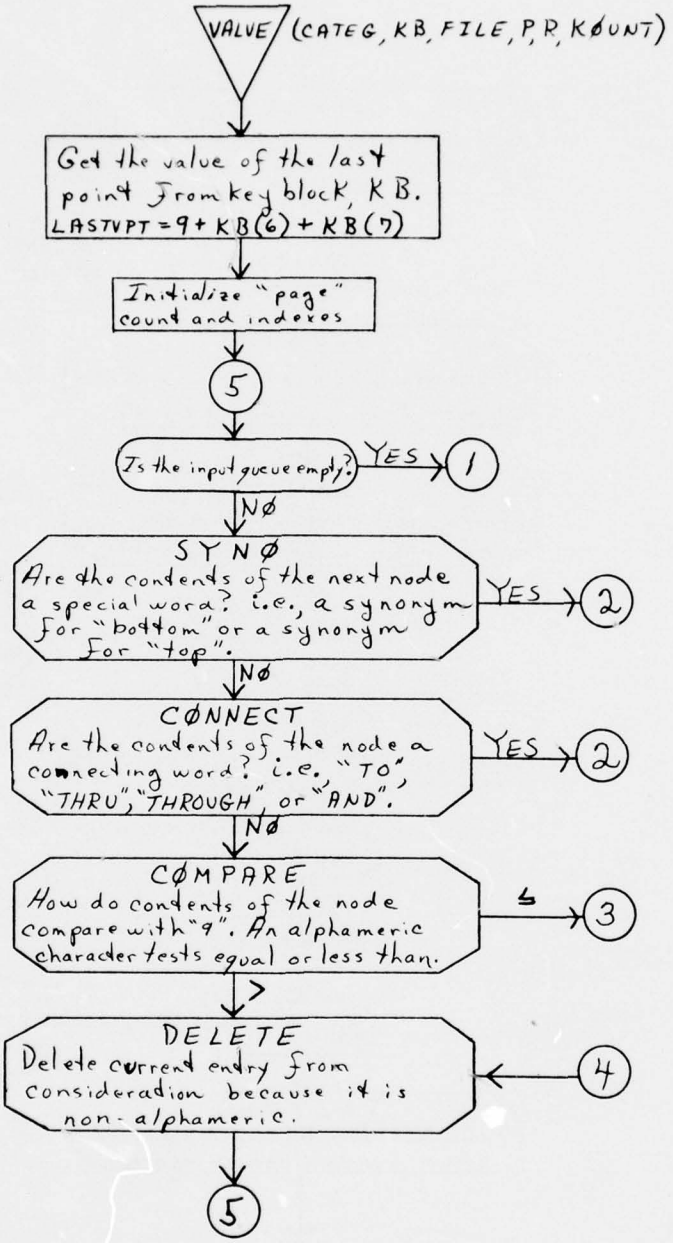
FDDEF (2, 1) ← FD(2, K) ~ HEADB (2, K)

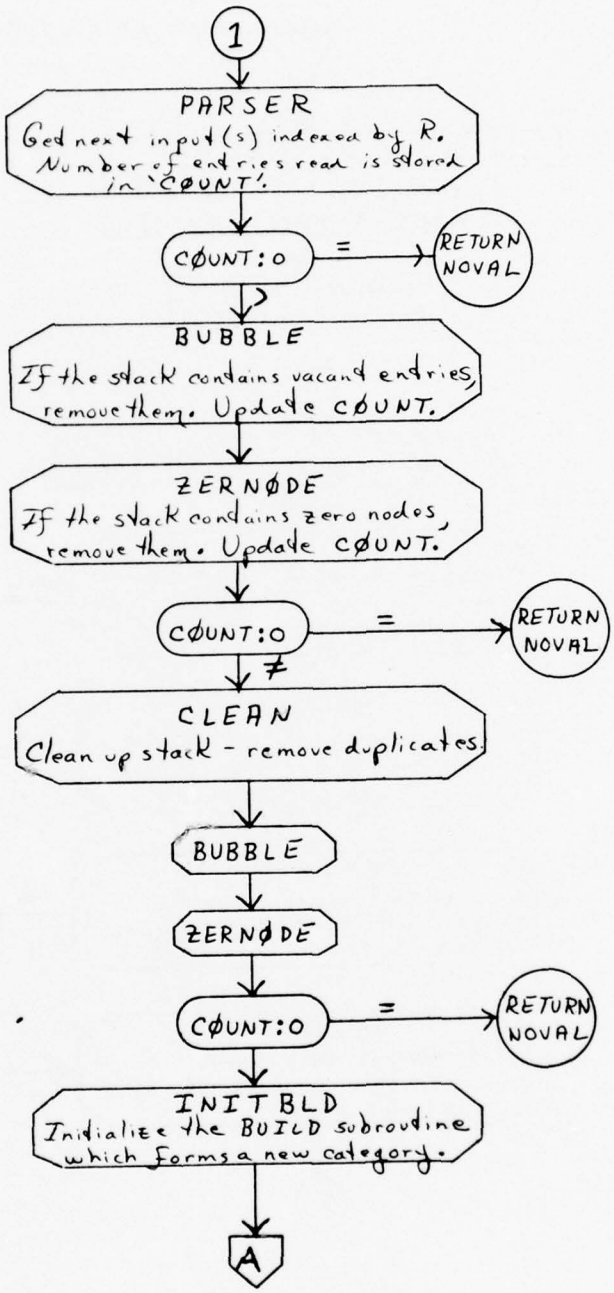
FDDEF (3, 1) ← FD(3, K) ~ HEADB (3, K)

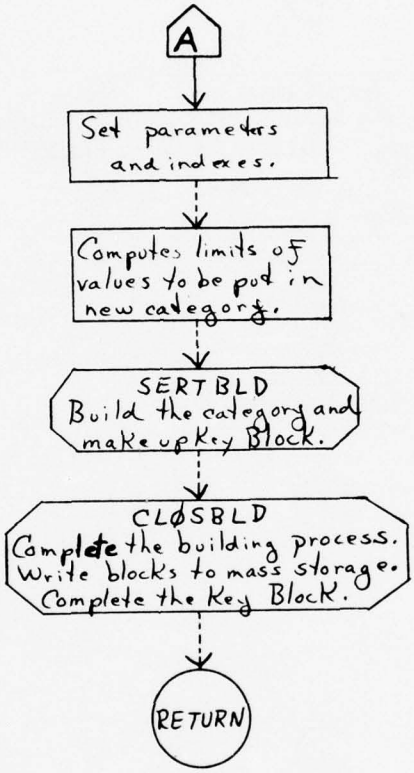
FDDEF (4, 1) ← FD(4, K) ~ HEADB (4, K)

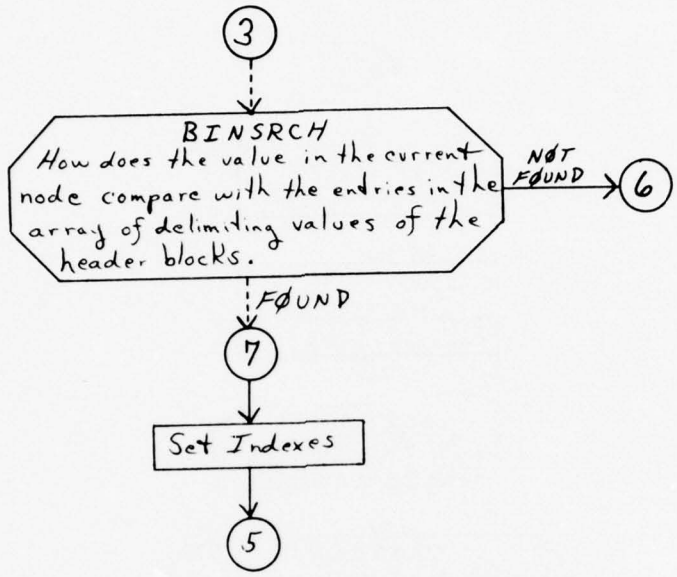
FDDEF (5, 1) ← FD(5, K) ~ HEADB (5, K)

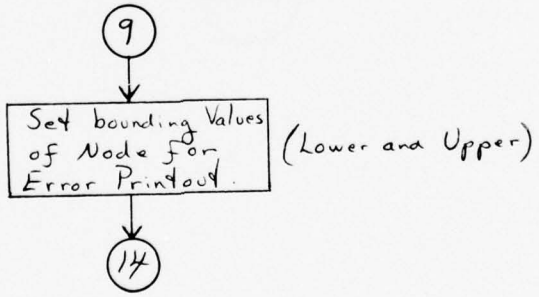

ØVERLAY(1,0)
VLØVL
ERR=.FALSE.
FIRST
.TRUE.
.FALSE.
Read Key Block 'IN' from file 'FILE' into KBB (64 words)
Read Key Block 'IN' from File 'FILE' into KBA (64 words)
Get values from input. VALUE(CATEG, KBB, FILE, R, S, COUNT)
Abnormal returns
Get values from input. VALUE(CATEG, FILE, R, S, COUNT)
2
4
2
STØP
ERR=.TRUE.

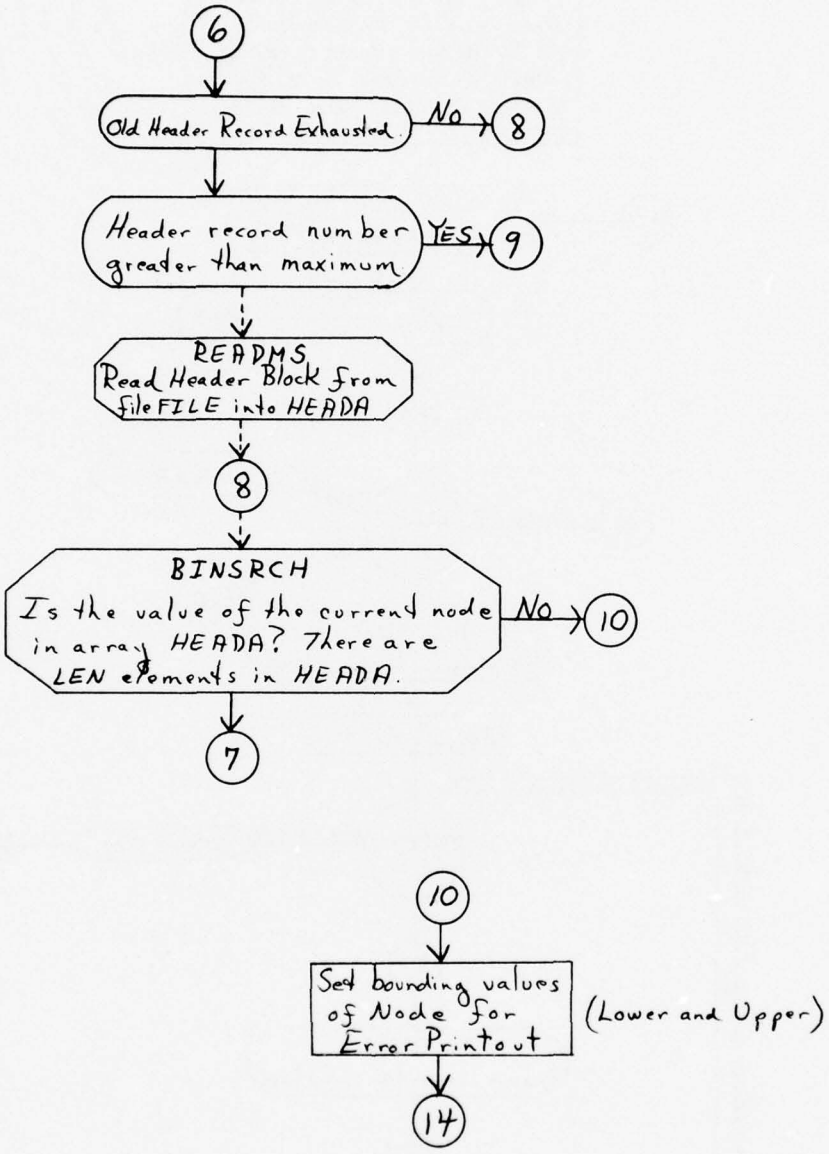

VALVE (CATEG, KB, FILE, P, R, KØUNT)
Get the value of the last point from key block, KB.
LASTVPT = 9 + KB(6) + KB(7)
Initialize "page" count and indexes
5
Is the input queue empty?
YES
1
NØ
SYNØ
Are the contents of the next node a special word? i.e., a synonym for "bottom" or a synonym for "top".
YES
2
NØ
CØNNECT
Are the contents of the node a connecting word? i.e., "TO", "THRU", "THROUGH", or "AND".
YES
2
NØ
CØMPARE
How do contents of the node compare with "9". An alphameric character tests equal or less than.
≤
3
>
DELETE
Delete current entry from consideration because it is non-alphameric.
4
5

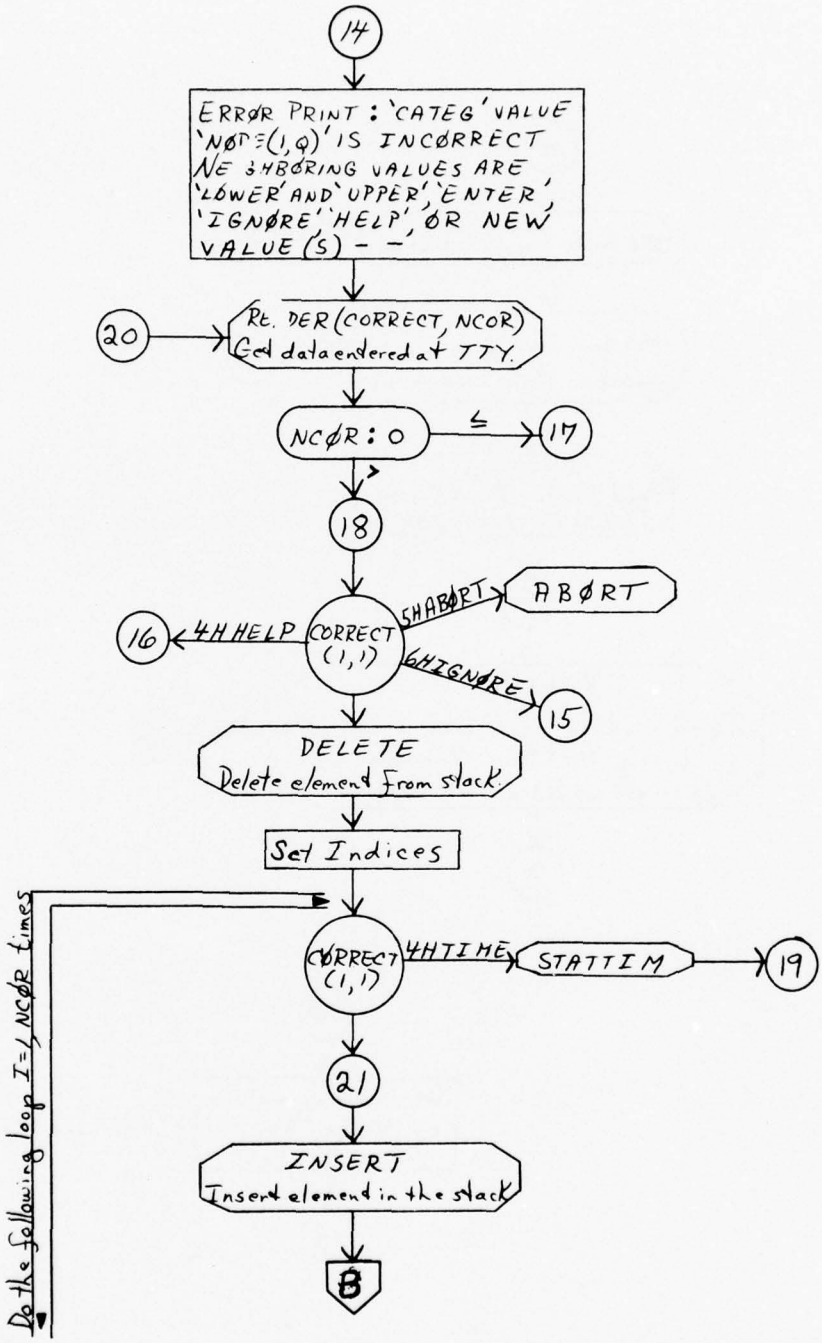

1
PARSER
Get next input(s) indexed by R. Number of entries read is stored in 'COUNT'.
COUNT:0
=
RETURN NOVAL
>
BUBBLE
If the stack contains vacant entries, remove them. Update COUNT.
ZERNODE
If the stack contains zero nodes, remove them. Update COUNT.
COUNT:0
=
RETURN NOVAL
≠
CLEAN
Clean up stack - remove duplicates
BUBBLE
ZERNODE
COUNT:0
=
RETURN NOVAL
INITBLD
Initialize the BUILD subroutine which forms a new category.
A

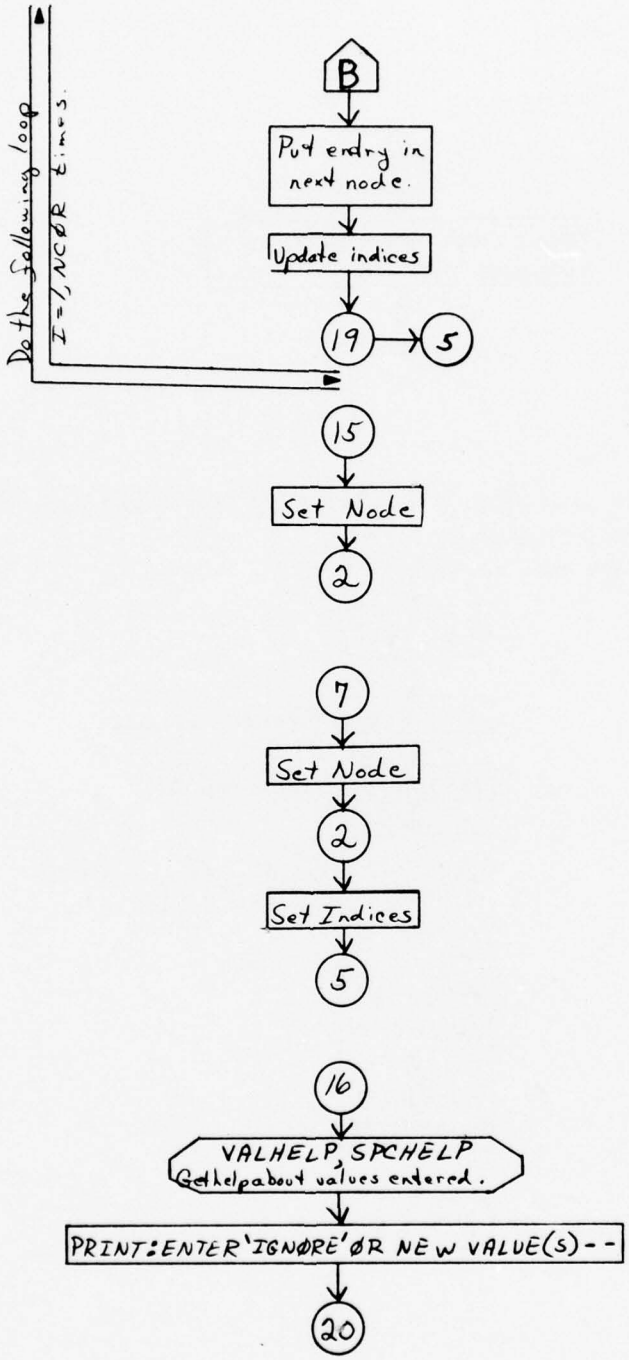

A
Set parameters and indexes.
Computes limits of values to be put in new category.
SERTBLD
Build the category and make up Key Block.
CLØSBLD
Complete the building process. Write blocks to mass storage. Complete the Key Block.
RETURN

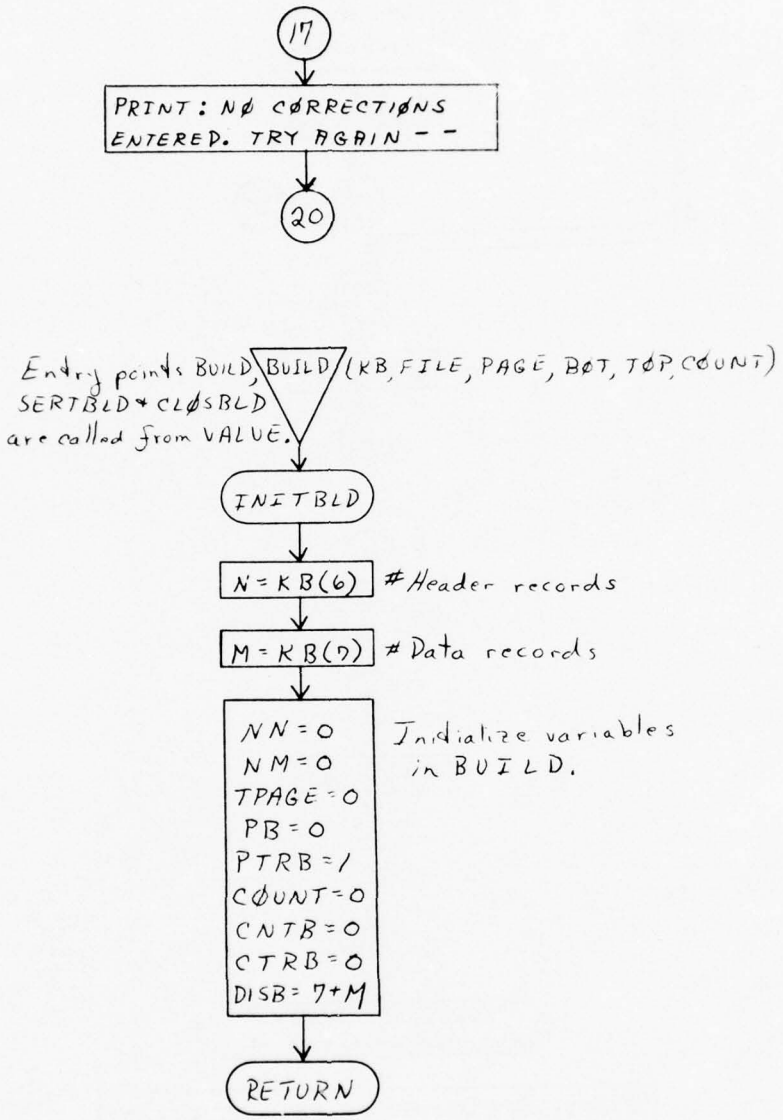

3
BINSRCH
How does the value in the current node compare with the entries in the array of delimiting values of the header blocks.
NOT FOUND
6
FOUND
7
Set Indexes
5

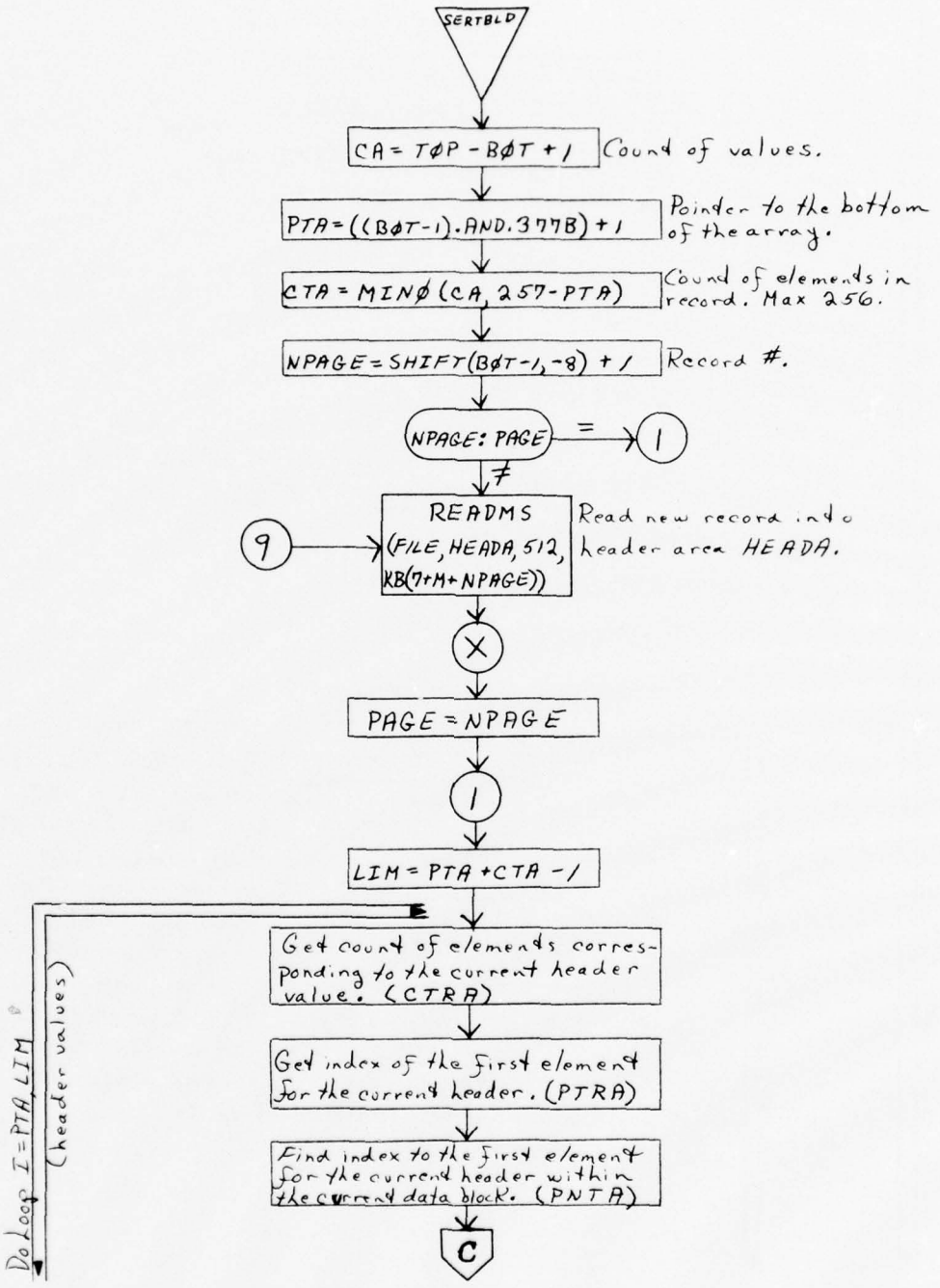

9
Set bounding Values of Node for Error Printout
(Lower and Upper)
14

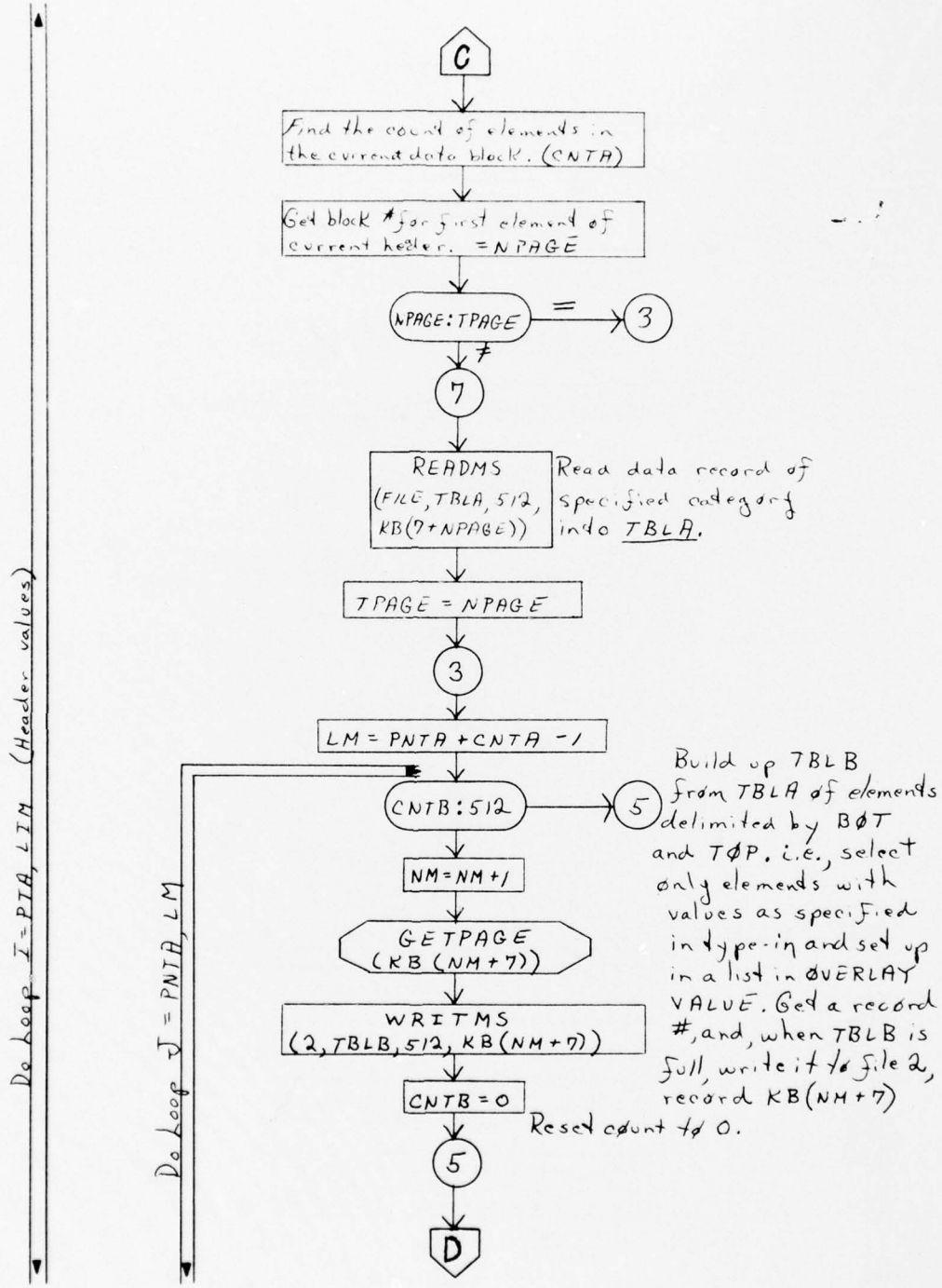

6
Old Header Record Exhausted.
No
8
Header record number greater than maximum
YES
9
READMS
Read Header Block from file FILE into HEADA
8
BINSRCH
Is the value of the current node in array HEADA? There are LEN elements in HEADA.
No
10
7
10
Set bounding values of Node for Error Printout
(Lower and Upper)
14

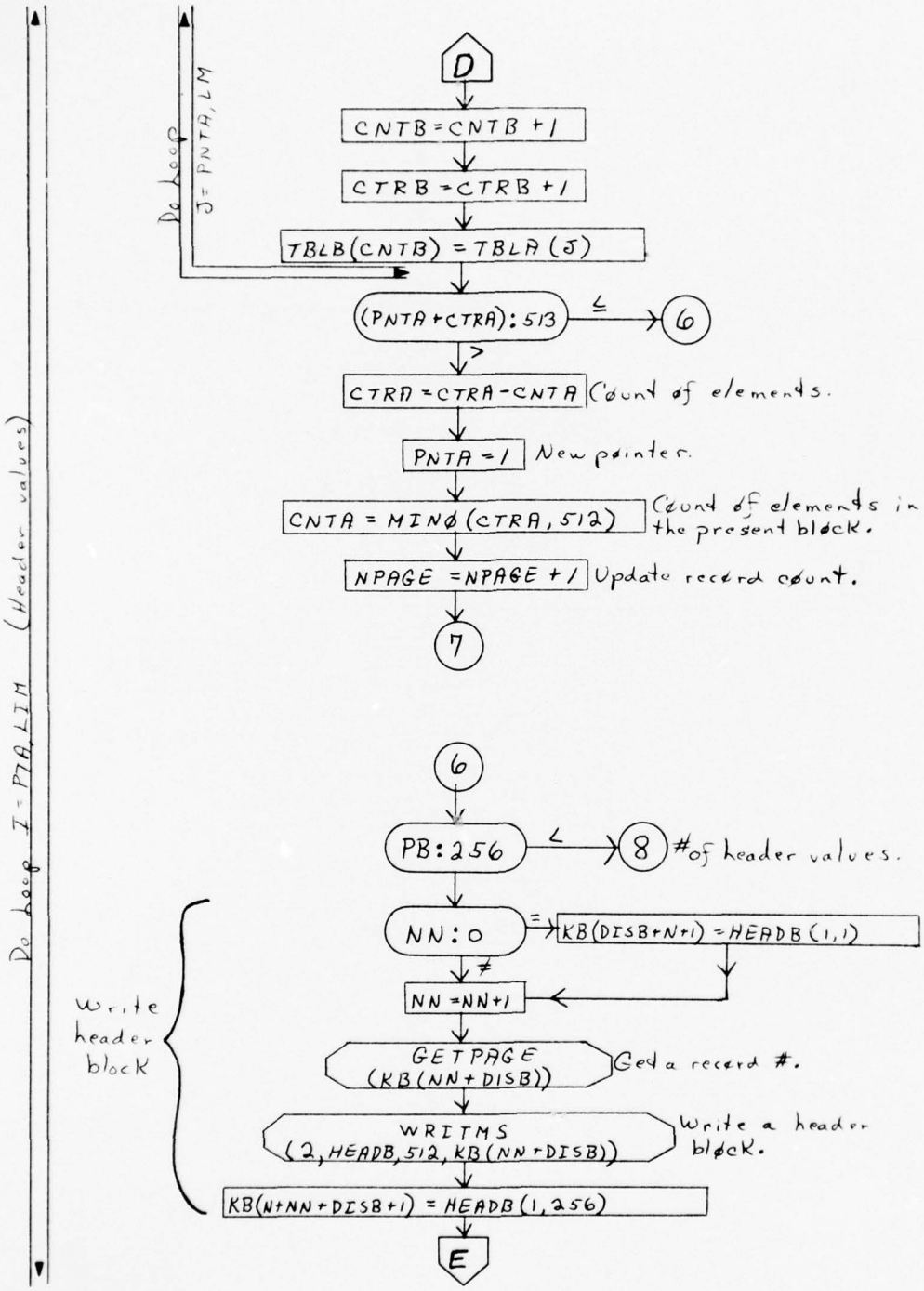

14
ERRØR PRINT : 'CATEG' VALUE 'NØT=(1,9)' IS INCØRRECT NE 3HBØRING VALUES ARE 'LØWER' AND 'UPPER', 'ENTER', 'IGNØRE', 'HELP', ØR NEW VALUE(S) - -
20
RE. DER(CORRECT, NCOR) Get data entered at TTY.
NCØR : 0
≤
17
>
18
CORRECT (1,1)
5HABØRT
ABØRT
16
4HHELP
6HIGNØRE
15
DELETE
Delete element from stack
Set Indices
CORRECT (1,1)
4HTIME
STATTIM
19
21
INSERT
Insert element in the stack
B
Do the following loop I=1, NCØR times

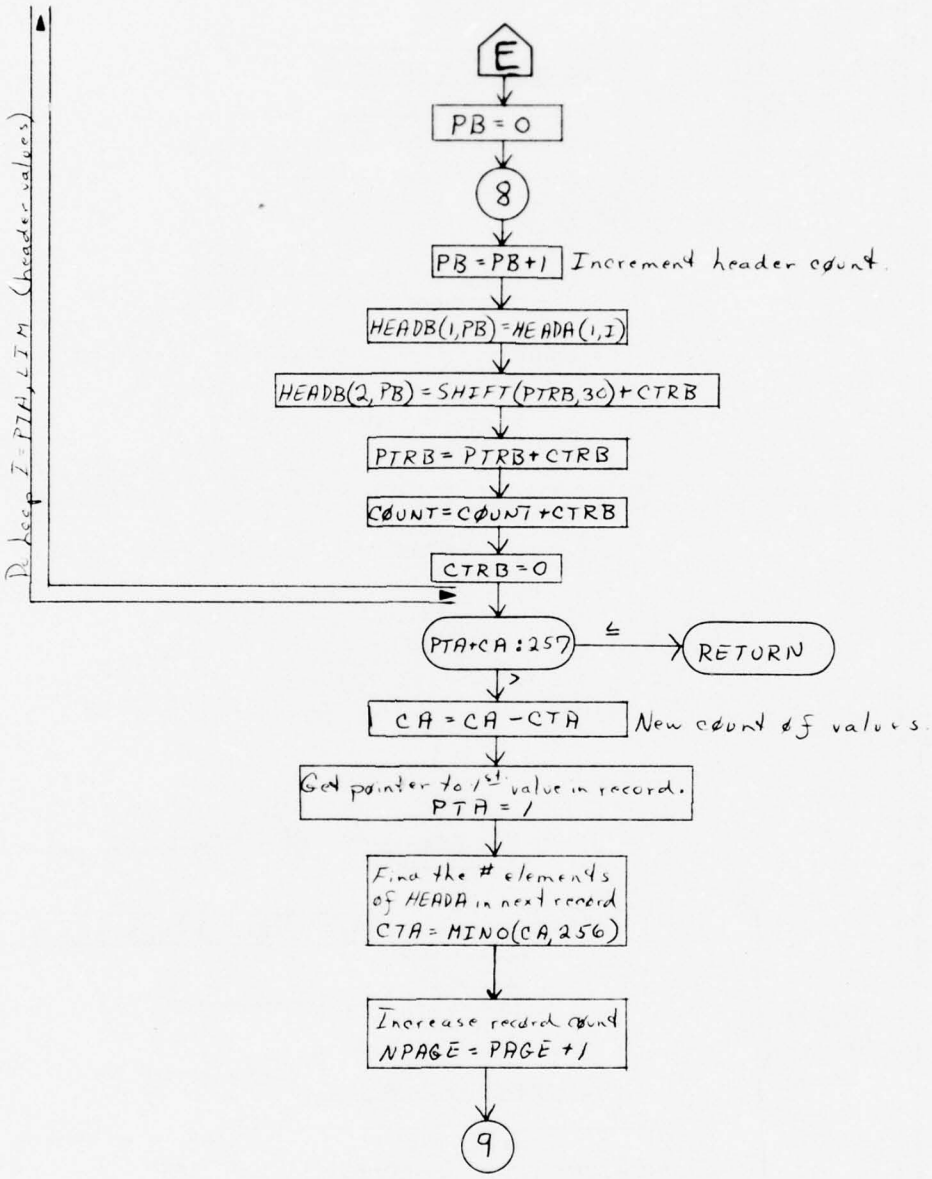

Do the following loop
I=1,NCØR times.
B
Put entry in next node.
Update indices
19
5
15
Set Node
2
7
Set Node
2
Set Indices
5
16
VALHELP, SPCHELP
Get help about values entered.
PRINT: ENTER 'IGNØRE' ØR NEW VALUE(S) - -
20

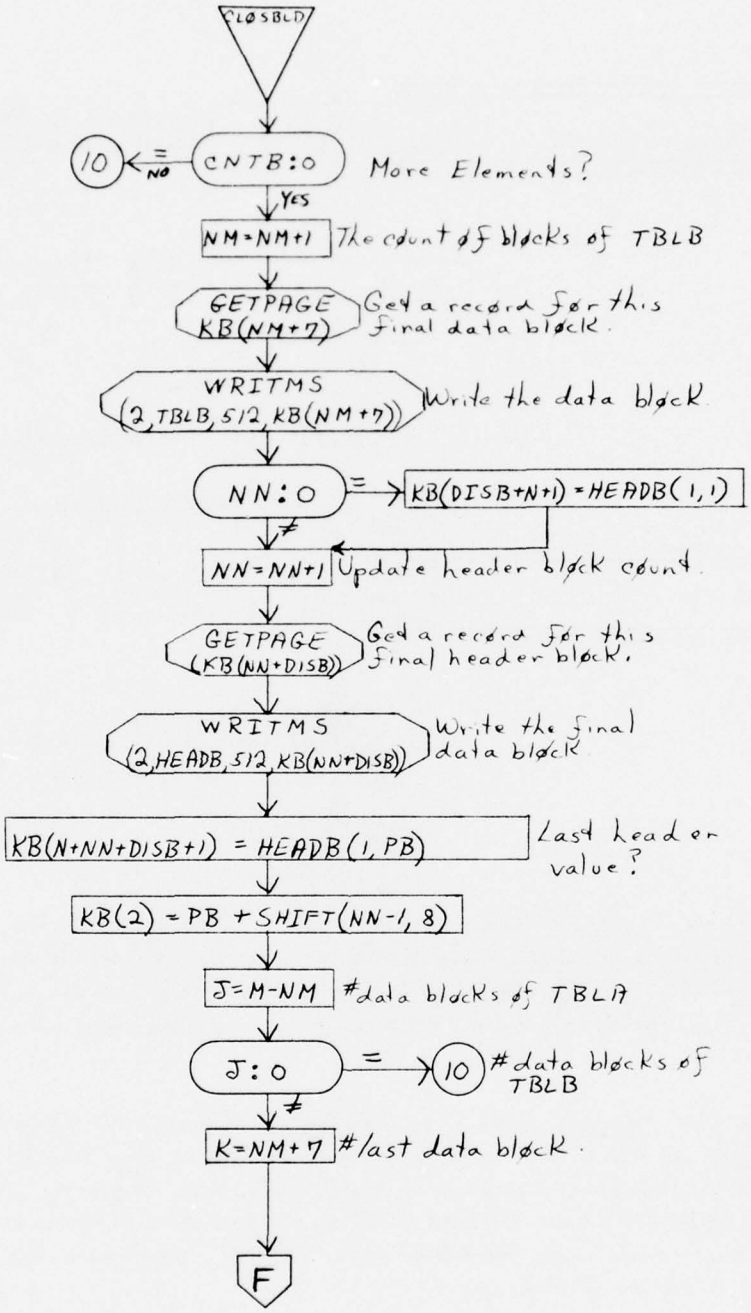

17
PRINT: NØ CØRRECTIØNS ENTERED. TRY AGAIN - -
20
Entry points BUILD, SERTBLD & CLØSBLD are called from VALUE.
BUILD (KB, FILE, PAGE, BØT, TØP, CØUNT)
INITBLD
N = KB(6)
# Header records
M = KB(7)
# Data records
NN = 0
NM = 0
TPAGE = 0
PB = 0
PTRB = 1
CØUNT = 0
CNTB = 0
CTRB = 0
DISB = 7+M
Initialize variables in BUILD.
RETURN

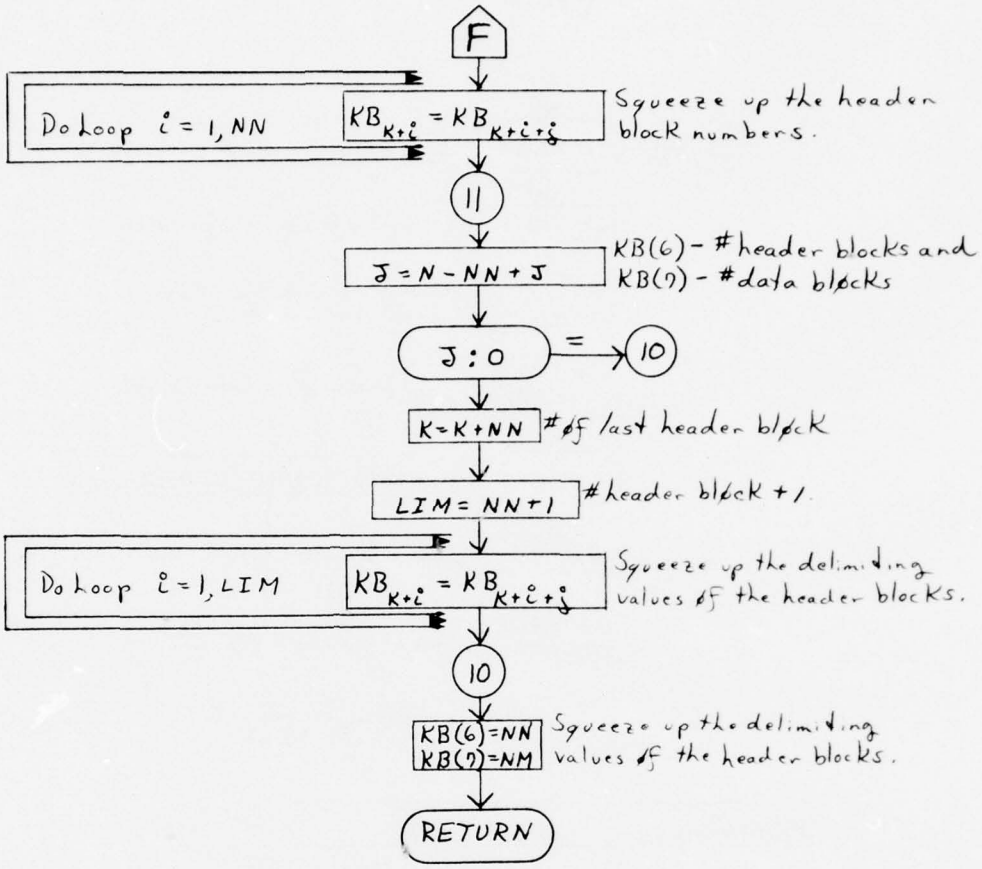

SERTBLD
CA = TØP - BØT + 1
Count of values.
PTA = ((BØT-1).AND.377B) + 1
Pointer to the bottom of the array.
CTA = MINØ(CA, 257-PTA)
Count of elements in record. Max 256.
NPAGE = SHIFT(BØT-1,-8) + 1
Record #.
NPAGE: PAGE
=
1
≠
9
READMS (FILE, HEADA, 512, KB(7+M+NPAGE))
Read new record into header area HEADA.
X
PAGE = NPAGE
1
LIM = PTA + CTA - 1
Do Loop I = PTA, LIM (header values)
Get count of elements corresponding to the current header value. (CTRA)
Get index of the first element for the current header. (PTRA)
Find index to the first element for the current header within the current data block. (PNTA)
C


Do Loop I = PTA, LIM (Header values)
C
Find the count of elements in the current data block. (CNTA)
Get block # for first element of current header. = NPAGE
NPAGE : TPAGE
=
3
≠
7
READMS (FILE, TBLA, 512, KB(7+NPAGE))
Read data record of specified category into TBLA.
TPAGE = NPAGE
3
LM = PNTA + CNTA - 1
Do Loop J = PNTA, LM
CNTB : 512
5
Build up TBLB from TBLA of elements delimited by BØT and TØP. i.e., select only elements with values as specified in type-in and set up in a list in ØVERLAY VALUE. Get a record #, and, when TBLB is full, write it to file 2, record KB(NM+7)
NM = NM + 1
GETPAGE (KB (NM+7))
WRITMS (2, TBLB, 512, KB(NM+7))
CNTB = 0
Reset count to 0.
5
D


Do loop I = PTA, LIM (Header values)
Do loop J = PNTA, LM
D
CNTB = CNTB + 1
CTRB = CTRB + 1
TBLB(CNTB) = TBLA(J)
(PNTA + CTRA) : 513
≤
6
>
CTRA = CTRA - CNTA
Count of elements.
PNTA = 1
New pointer.
CNTA = MIN0(CTRA, 512)
Count of elements in the present block.
NPAGE = NPAGE + 1
Update record count.
7
6
PB : 256
<
8
# of header values.
NN : 0
=
KB(DISB + N + 1) = HEADB(1, I)
≠
NN = NN + 1
Write header block
GETPAGE (KB(NN + DISB))
Get a record #.
WRITMS (2, HEADB, 512, KB(NN + DISB))
Write a header block.
KB(N + NN + DISB + 1) = HEADB(1, 256)
E

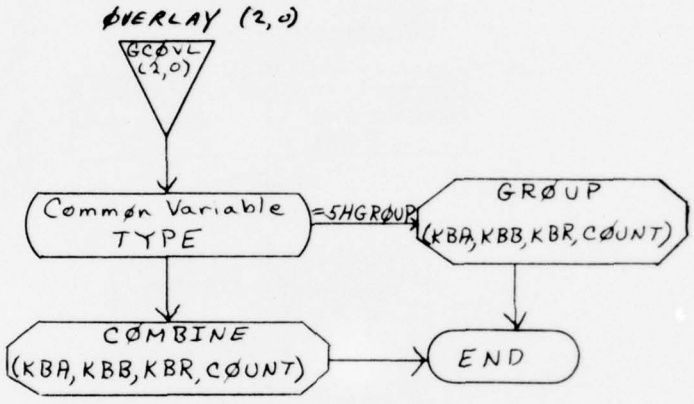

E
PB = 0
8
PB = PB+1
Increment header count
HEADB(1,PB) = HEADA(1,I)
HEADB(2,PB) = SHIFT(PTRB,30)+CTRB
PTRB = PTRB+CTRB
COUNT = COUNT+CTRB
CTRB = 0
Loop I = PTH,LIM (header values)
PTA+CA : 257
≤
RETURN
>
CA = CA - CTA
New count of values
Get pointer to 1st value in record.
PTA = 1
Find the # elements of HEADA in next record
CTA = MIN0(CA,256)
Increase record count
NPAGE = PAGE+1
9

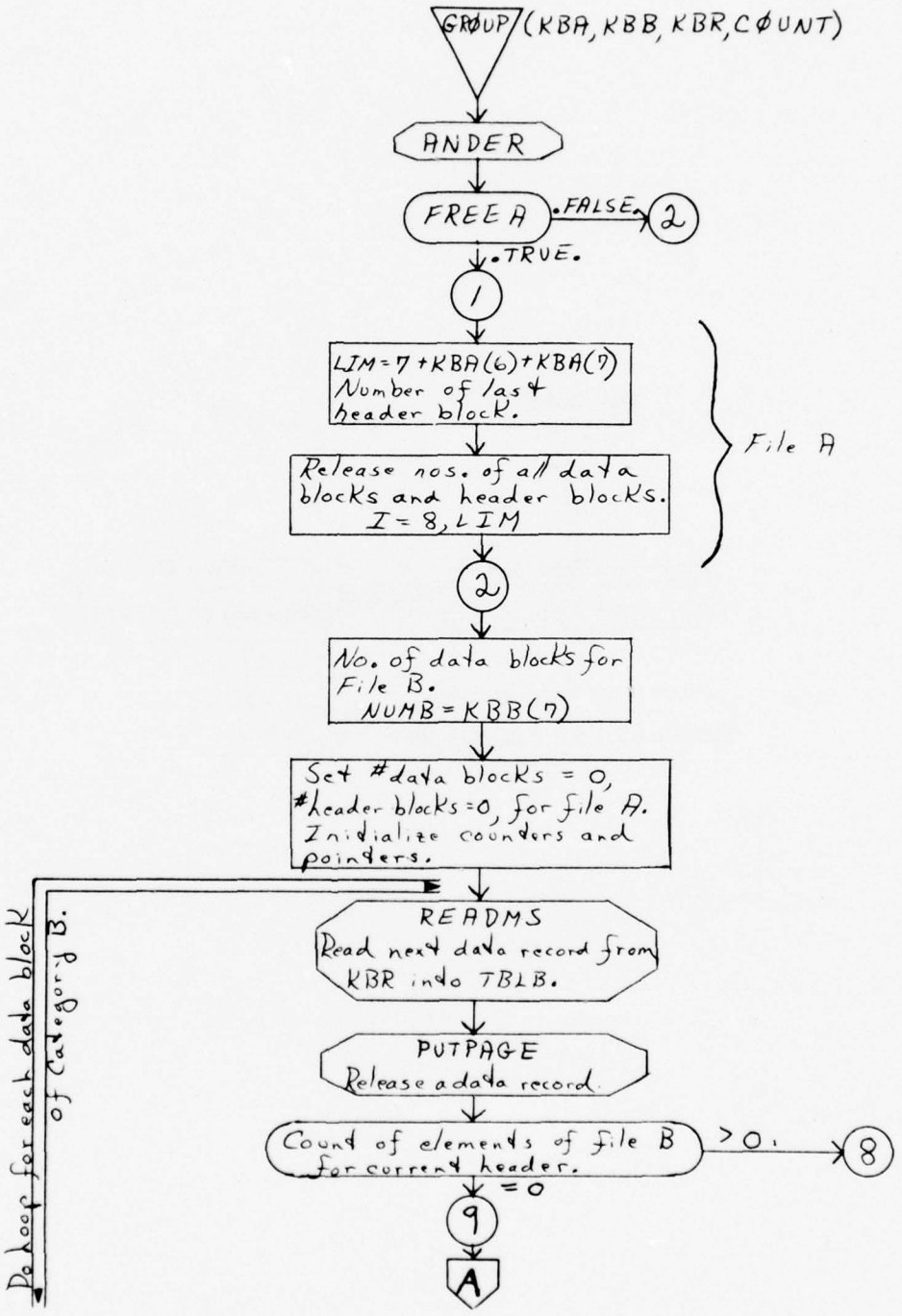

CLOSBLD
10
= NO
CNTB:0
More Elements?
Yes
NM=NM+1
The count of blocks of TBLB
GETPAGE KB(NM+7)
Get a record for this final data block.
WRITMS (2,TBLB,512,KB(NM+7))
Write the data block
NN:0
=
KB(DISB+N+1)=HEADB(1,1)
≠
NN=NN+1
Update header block count
GETPAGE (KB(NN+DISB))
Get a record for this final header block.
WRITMS (2,HEADB,512,KB(NN+DISB))
Write the final data block
KB(N+NN+DISB+1) = HEADB(1,PB)
Last header value?
KB(2) = PB + SHIFT(NN-1,8)
J=M-NM
# data blocks of TBLA
J:0
=
10
data blocks of TBLB
≠
K=NM+7
# last data block.
F

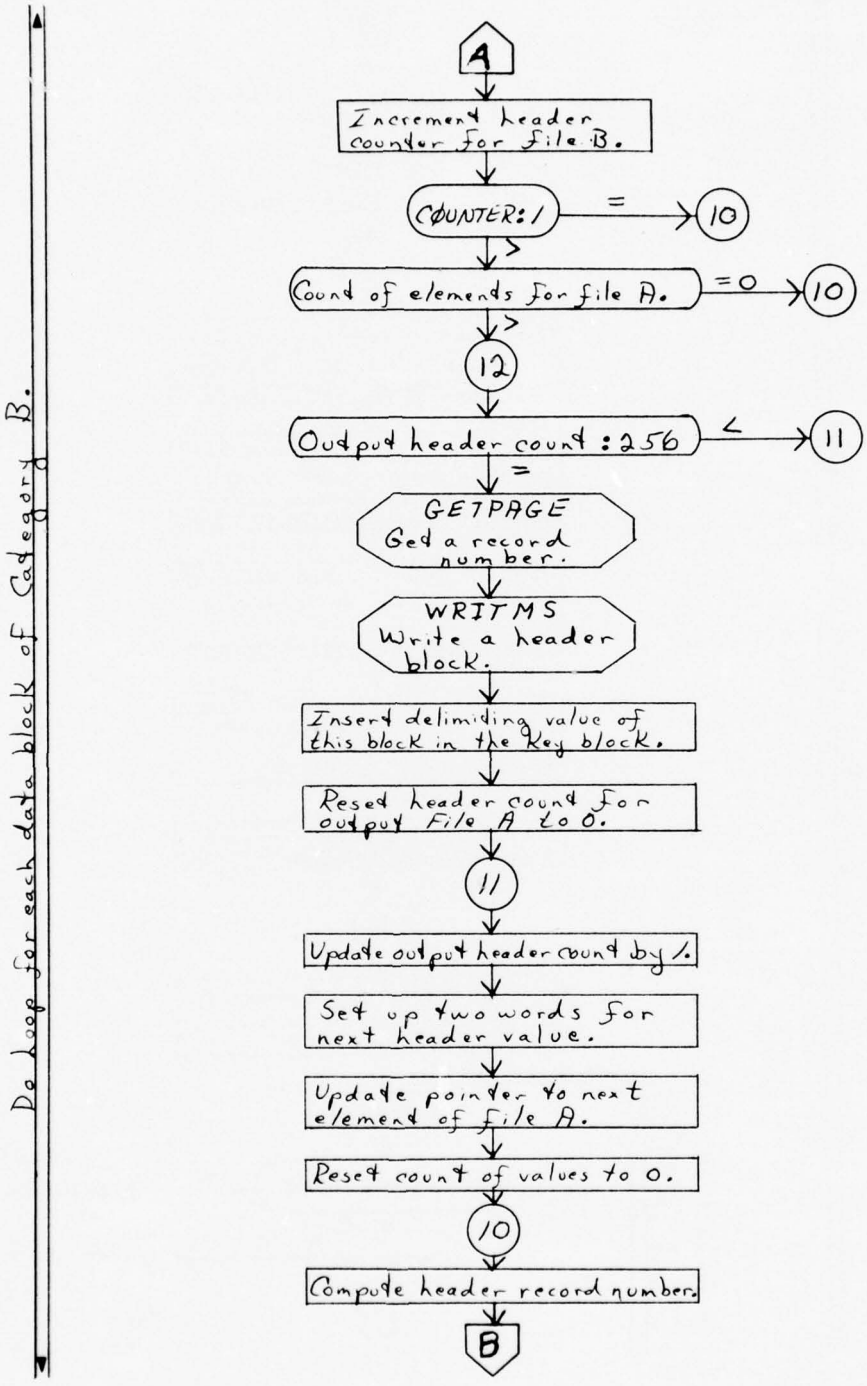


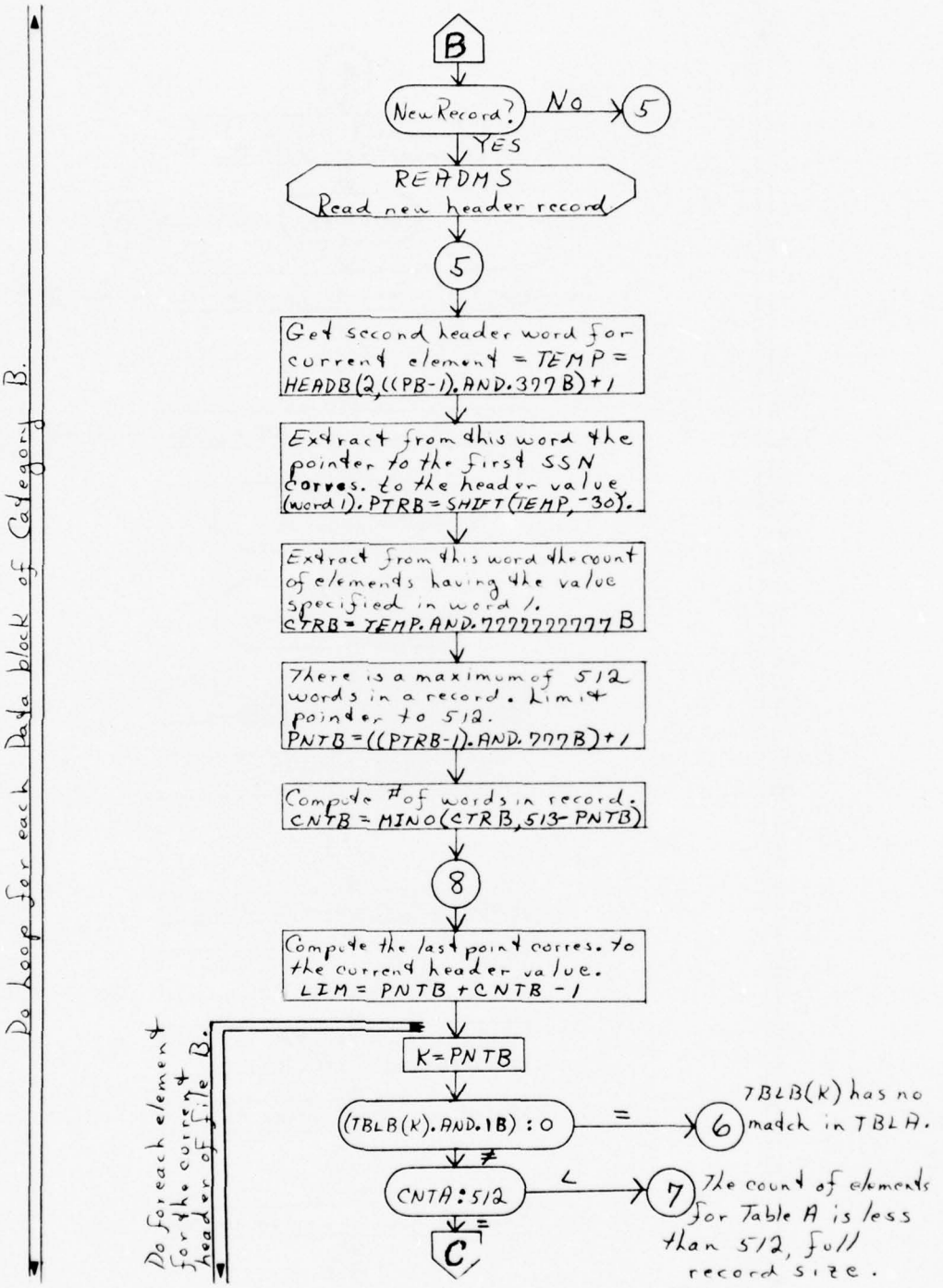
(1) When finished here, KB ≡ KBA is the Key block. The <u>heada</u> and <u>data</u> blocks are written to disk and referenced by <u>KBA</u>. In the process of writing these blocks to disk, HEADB(512) and TBLB(512) have been used.

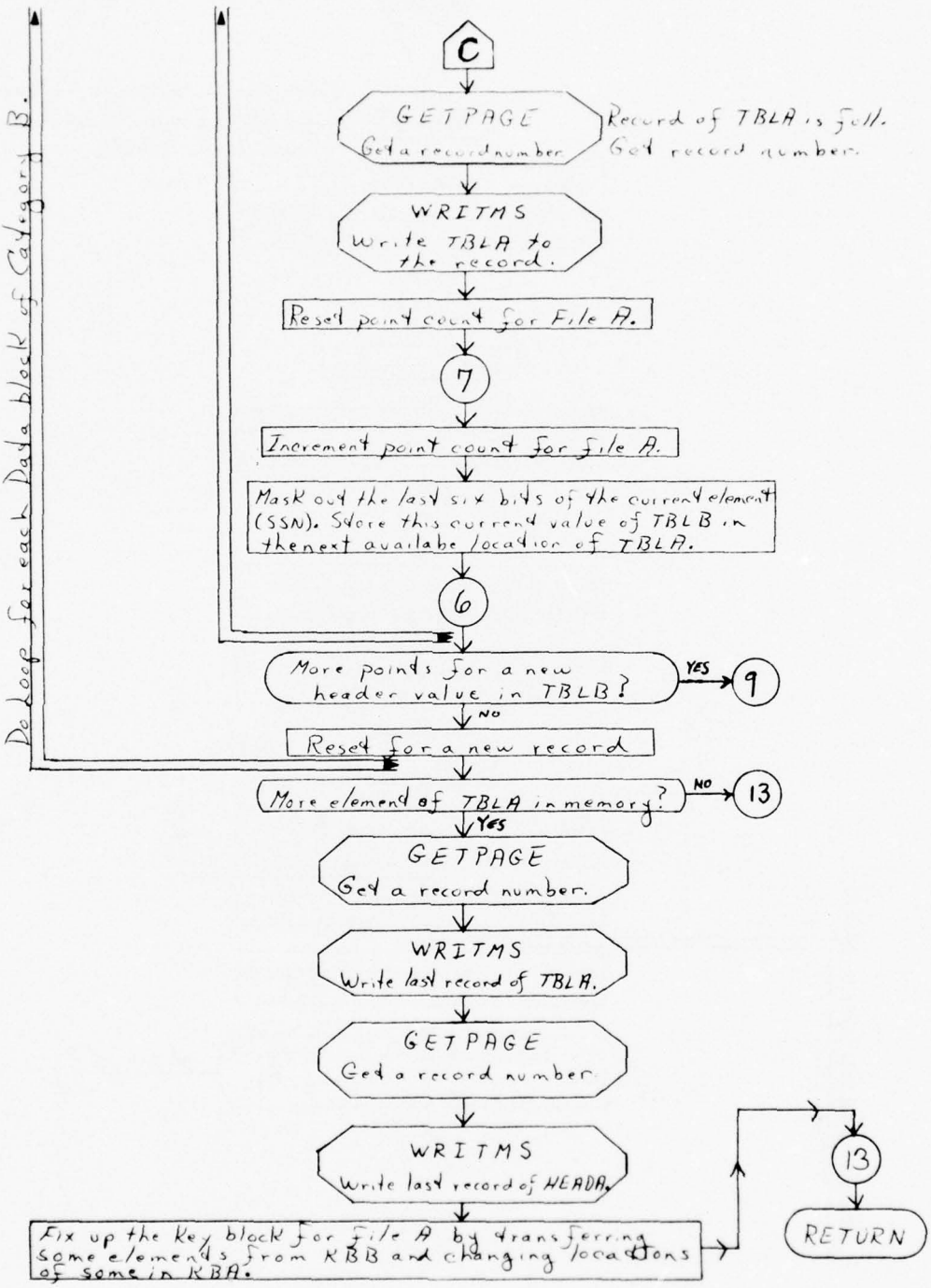
(2) On the <u>second</u> entry to <u>VLOVL</u>, KB is set equal to <u>KBB</u> so that this array is set up as a Key block for the assembled header and data blocks. Again, HEADB(512) and TBLB(512) are used to assemble the data read into HEADA and TBLA, and are written to disk as filled up.

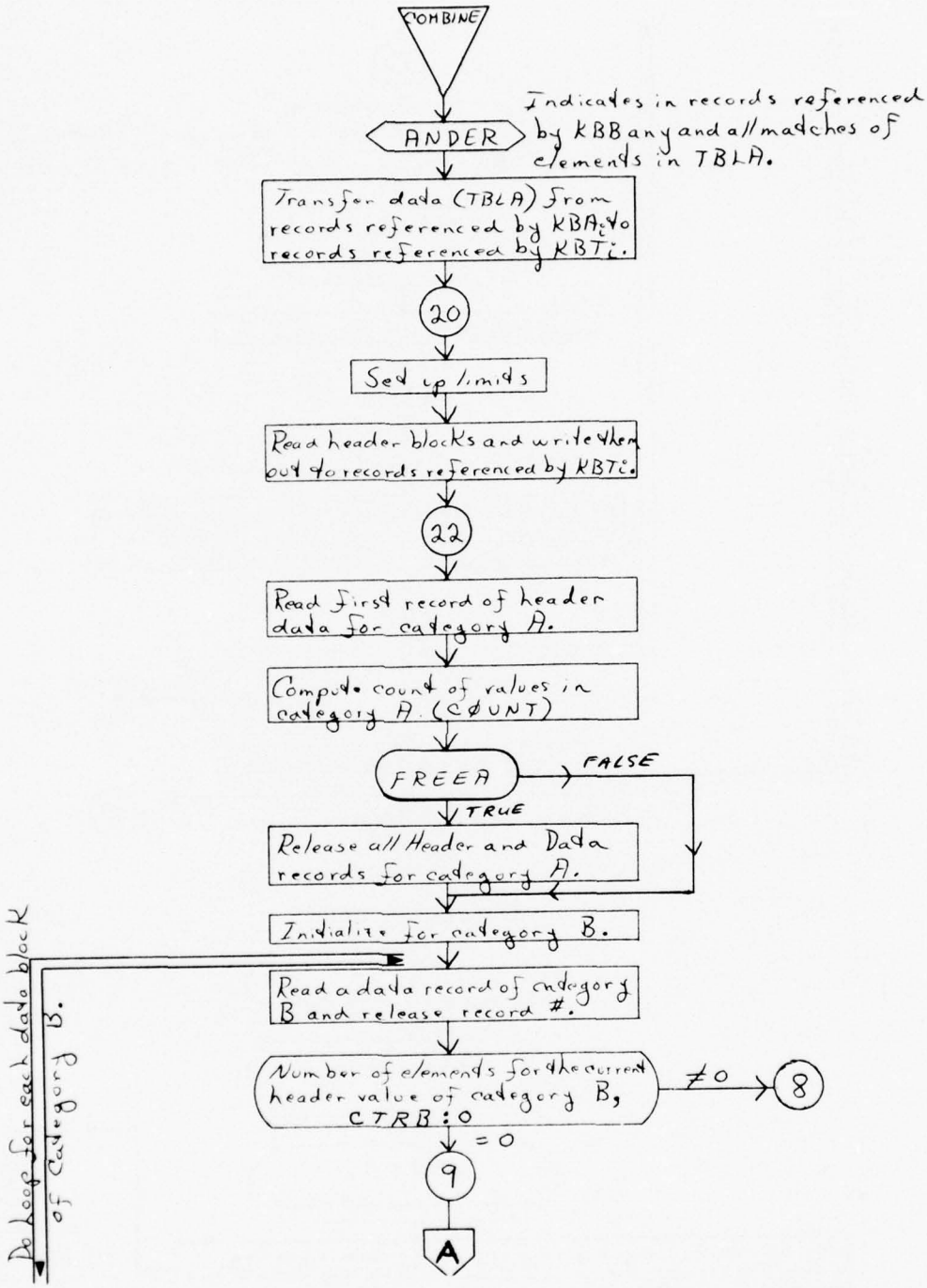

ØVERLAY (2,0)
GCØVL (2,0)
Commøn Variable TYPE
=5HGRØUP
GRØUP (KBA,KBB,KBR,CØUNT)
CØMBINE (KBA,KBB,KBR,CØUNT)
END

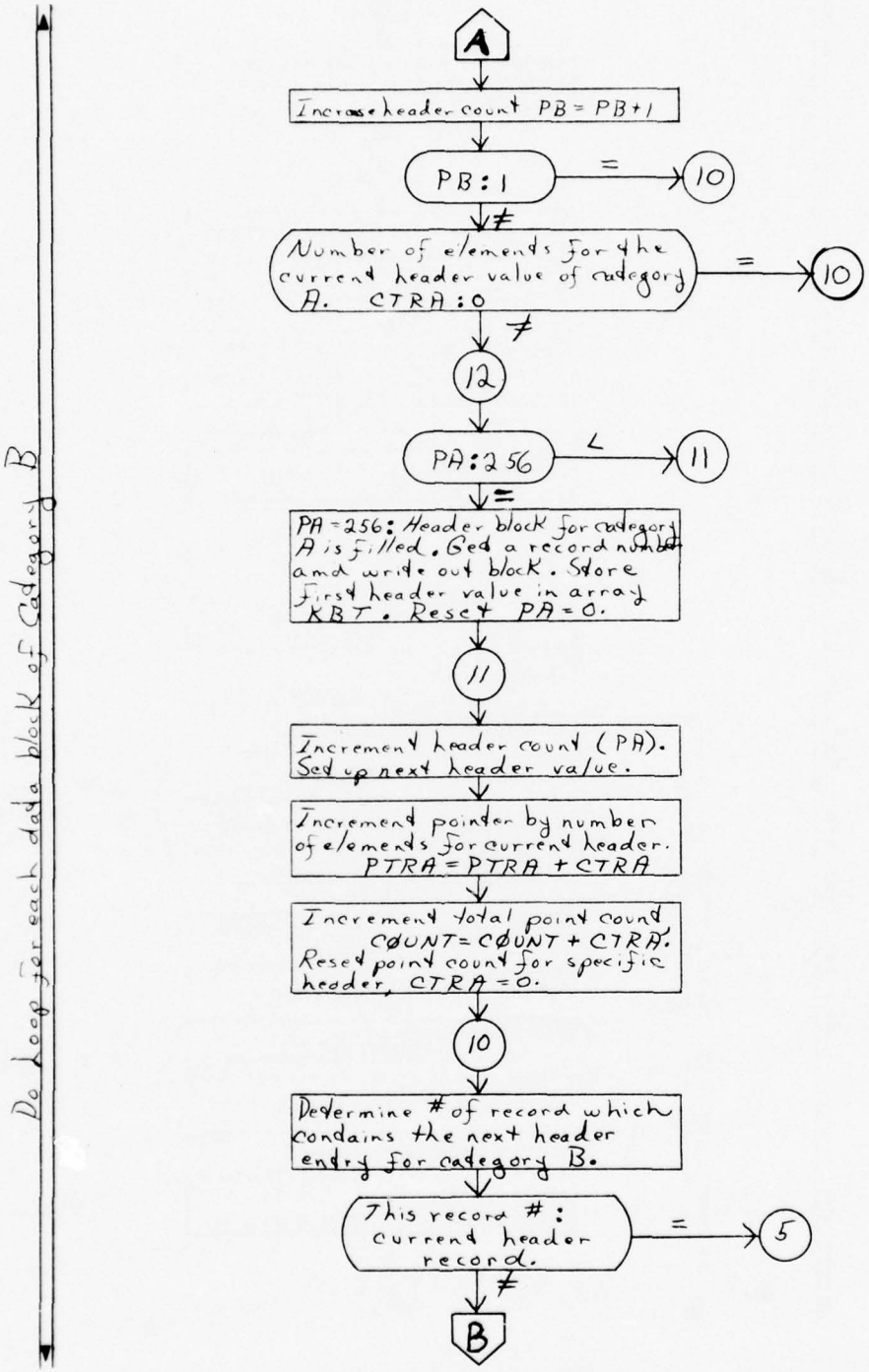

GROUP (KBA, KBB, KBR, COUNT)
ANDER
FREE A
.FALSE.
2
.TRUE.
1
LIM = 7 + KBA(6) + KBA(7)
Number of last header block.
Release nos. of all data blocks and header blocks.
I = 8, LIM
File A
2
No. of data blocks for File B.
NUMB = KBB(7)
Set # data blocks = 0, # header blocks = 0, for file A.
Initialize counters and pointers.
Do loop for each data block of category B.
READMS
Read next data record from KBR into TBLB.
PUTPAGE
Release a data record.
Count of elements of file B for current header.
> 0.
8
= 0
9
A

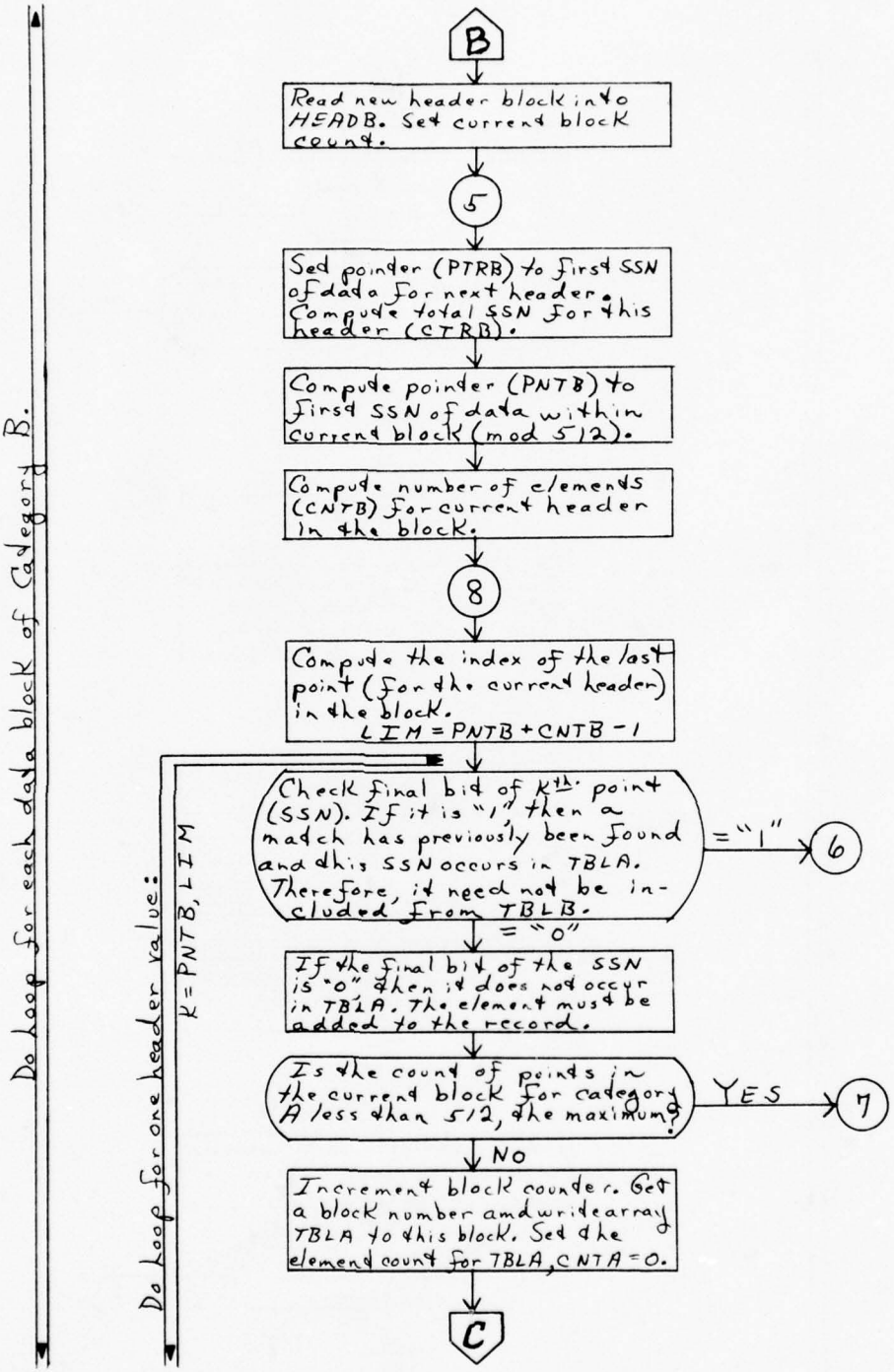

Do loop for each data block of Category B.
A
Increment header counter for file B.
COUNTER: 1
=
10
>
Count of elements for file A.
=0
10
>
12
Output header count : 256
<
11
=
GETPAGE
Get a record number.
WRITMS
Write a header block.
Insert delimiting value of this block in the key block.
Reset header count for output File A to 0.
11
Update output header count by 1.
Set up two words for next header value.
Update pointer to next element of file A.
Reset count of values to 0.
10
Compute header record number.
B

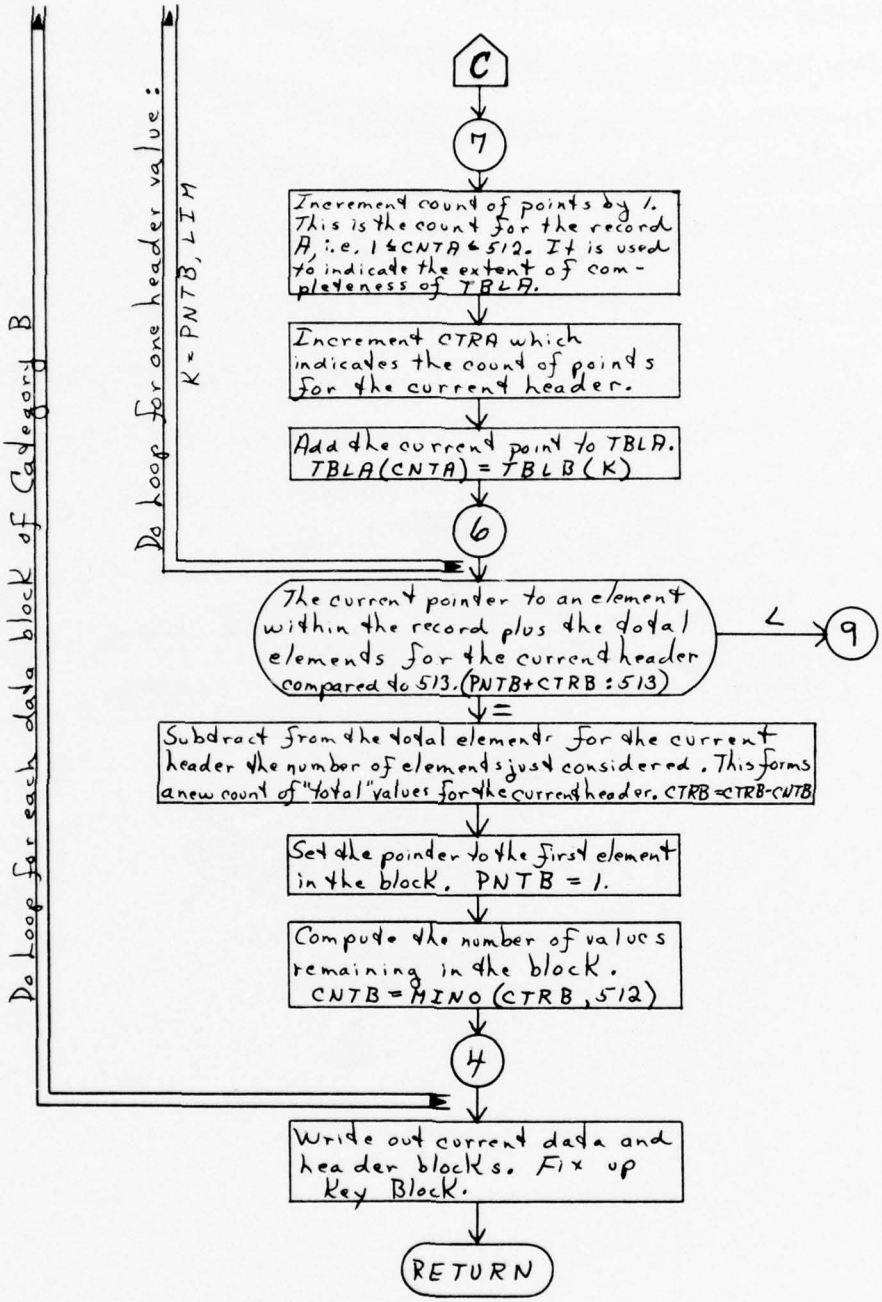

Do loop for each Data block of Category B.
B
New Record?
No
5
YES
READMS
Read new header record
5
Get second header word for current element = TEMP = HEADB(2,((PB-1).AND.377B)+1
Extract from this word the pointer to the first SSN corres. to the header value (word 1). PTRB = SHIFT(TEMP,-30).
Extract from this word the count of elements having the value specified in word 1.
CTRB = TEMP.AND.777777777777B
There is a maximum of 512 words in a record. Limit pointer to 512.
PNTB = ((PTRB-1).AND.777B)+1
Compute # of words in record.
CNTB = MIN0(CTRB, 513-PNTB)
8
Compute the last point corres. to the current header value.
LIM = PNTB + CNTB - 1
Do for each element for the current header of file B.
K = PNTB
(TBLB(K).AND.1B) : 0
=
6
TBLB(K) has no match in TBLA.
≠
CNTA : 512
<
7
The count of elements for Table A is less than 512, full record size.
=
C

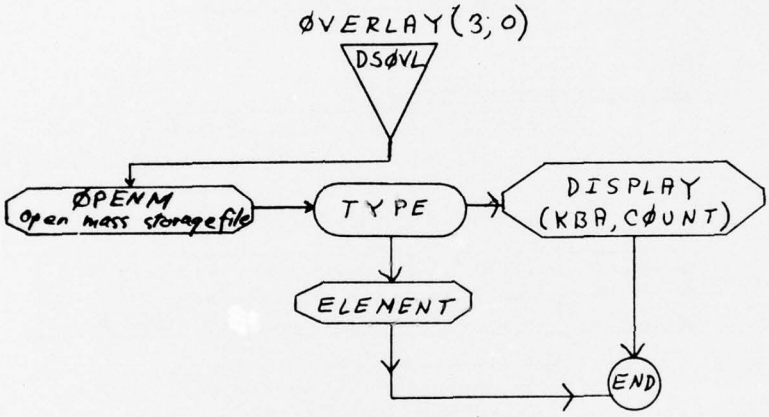

C
GETPAGE
Get a record number
Record of TBLA is full.
Get record number.
WRITMS
Write TBLA to the record.
Reset point count for File A.
7
Increment point count for file A.
Mask out the last six bits of the current element (SSN). Store this current value of TBLB in the next available location of TBLA.
6
More points for a new header value in TBLB?
YES
9
NO
Reset for a new record
More element of TBLA in memory?
NO
13
YES
GETPAGE
Get a record number.
WRITMS
Write last record of TBLA.
GETPAGE
Get a record number
WRITMS
Write last record of HEADA.
Fix up the key block for File A by transferring some elements from KBB and changing locations of some in KBA.
13
RETURN
Do loop for each Data block of Category B.

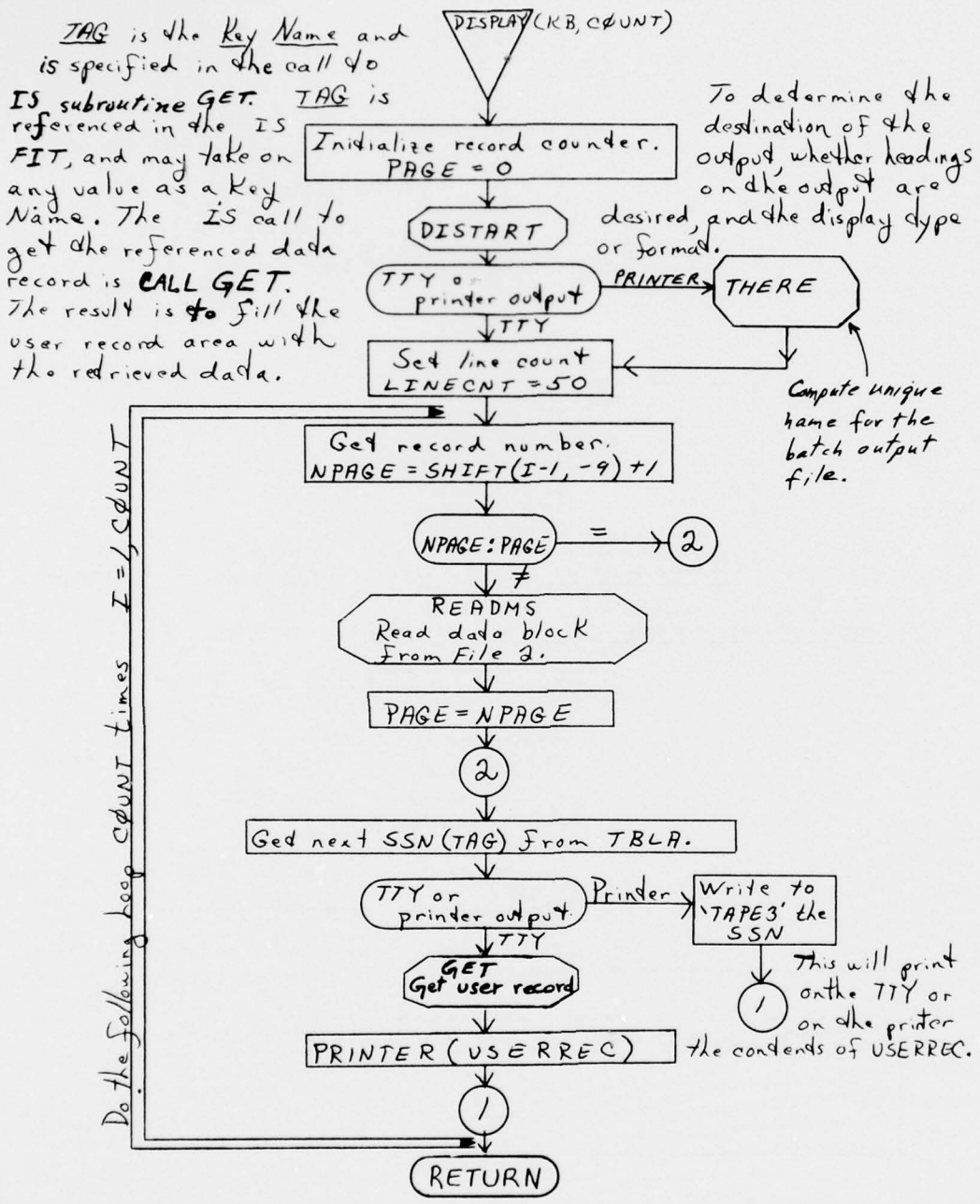

COMBINE
ANDER
Indicates in records referenced by KBB any and all matches of elements in TBLA.
Transfer data (TBLA) from records referenced by KBA; to records referenced by KBT;.
20
Set up limits
Read header blocks and write them out to records referenced by KBT;.
22
Read first record of header data for category A.
Compute count of values in category A. (COUNT)
FREEA
FALSE
TRUE
Release all Header and Data records for category A.
Initialize for category B.
Read a data record of category B and release record #.
Number of elements for the current header value of category B, CTRB:0
≠0
8
=0
9
A
Do loop for each data block of category B.

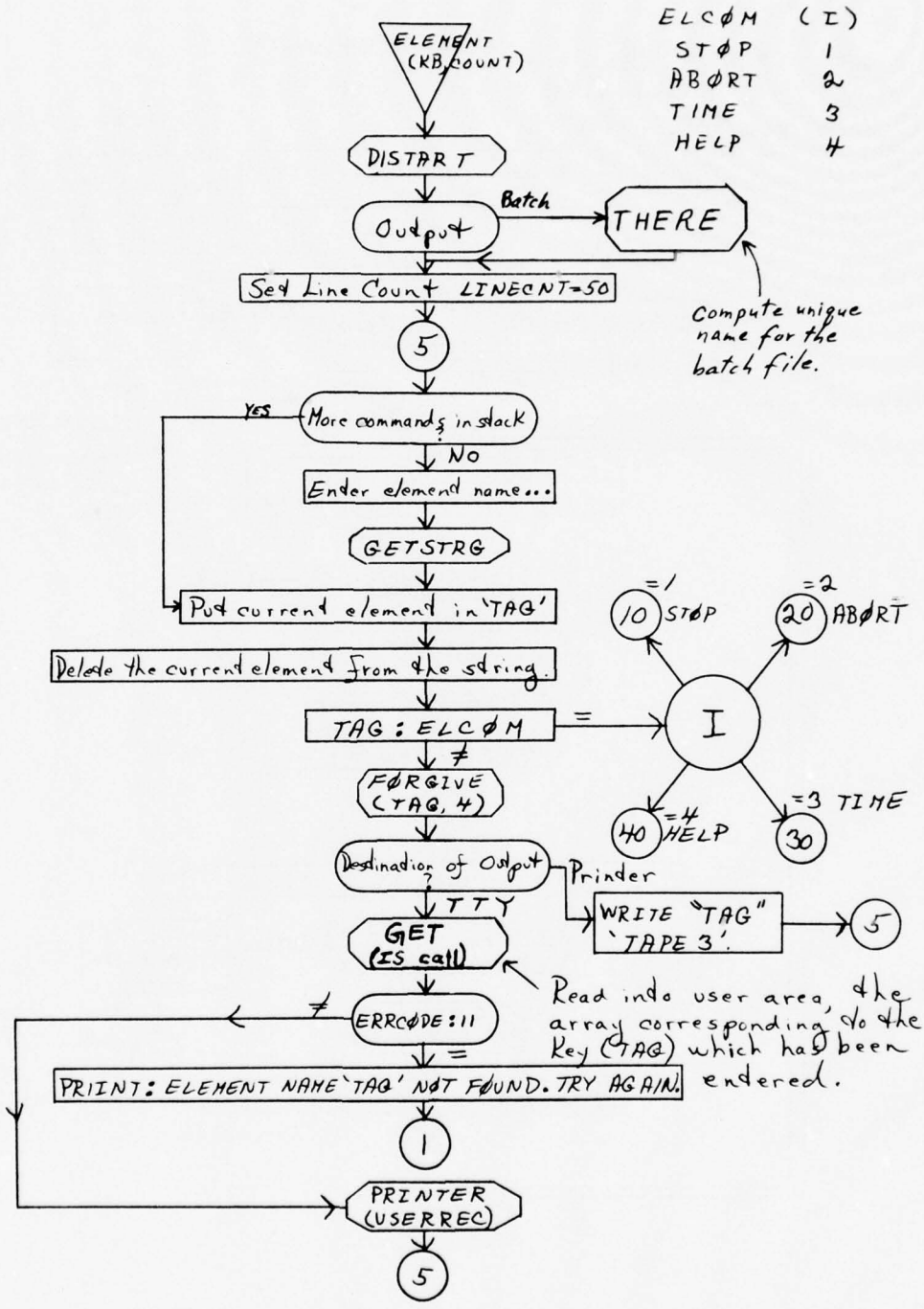

Do loop for each data block of category B
A
Increase header count PB = PB+1
PB : 1
=
10
≠
Number of elements for the current header value of category A. CTRA : 0
=
10
≠
12
PA : 256
<
11
=
PA = 256: Header block for category A is filled. Get a record number and write out block. Store first header value in array KBT. Reset PA = 0.
11
Increment header count (PA). Set up next header value.
Increment pointer by number of elements for current header. PTRA = PTRA + CTRA
Increment total point count COUNT = COUNT + CTRA. Reset point count for specific header, CTRA = 0.
10
Determine # of record which contains the next header entry for category B.
This record # : current header record.
=
5
≠
B

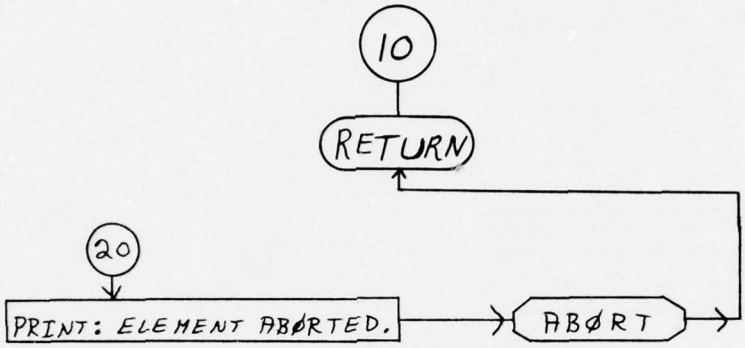

Do loop for each data block of category B.
B
Read new header block into HEADB. Set current block count.
5
Set pointer (PTRB) to first SSN of data for next header. Compute total SSN for this header (CTRB).
Compute pointer (PNTB) to first SSN of data within current block (mod 512).
Compute number of elements (CNTB) for current header in the block.
8
Compute the index of the last point (for the current header) in the block.
LIM = PNTB + CNTB - 1
Do loop for one header value:
K = PNTB, LIM
Check final bit of K^th point (SSN). If it is "1" then a match has previously been found and this SSN occurs in TBLA. Therefore, it need not be included from TBLB.
= "1"
6
= "0"
If the final bit of the SSN is "0" then it does not occur in TBLA. The element must be added to the record.
Is the count of points in the current block for category A less than 512, the maximum?
YES
7
NO
Increment block counter. Get a block number and write array TBLA to this block. Set the element count for TBLA, CNTA = 0.
C

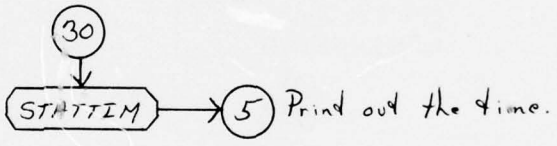

Do Loop for each data block of Category B
Do Loop for one header value:
K = PNTB, LIM
C
7
Increment count of points by 1. This is the count for the record A, i.e. 1 ≤ CNTA ≤ 512. It is used to indicate the extent of completeness of TBLA.
Increment CTRA which indicates the count of points for the current header.
Add the current point to TBLA. TBLA(CNTA) = TBLB(K)
6
The current pointer to an element within the record plus the total elements for the current header compared to 513. (PNTB+CTRB : 513)
<
9
=
Subtract from the total elements for the current header the number of elements just considered. This forms a new count of "total" values for the current header. CTRB = CTRB-CNTB
Set the pointer to the first element in the block. PNTB = 1.
Compute the number of values remaining in the block. CNTB = MINO (CTRB, 512)
4
Write out current data and header blocks. Fix up Key Block.
RETURN

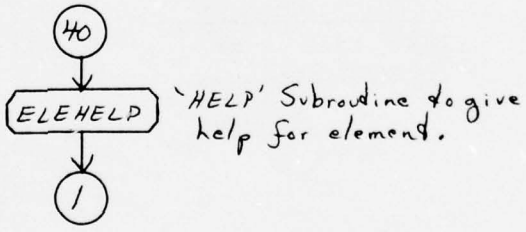

ØVERLAY(3,0)
DSØVL
ØPENM
open mass storage file
TYPE
DISPLAY
(KBA,CØUNT)
ELEMENT
END

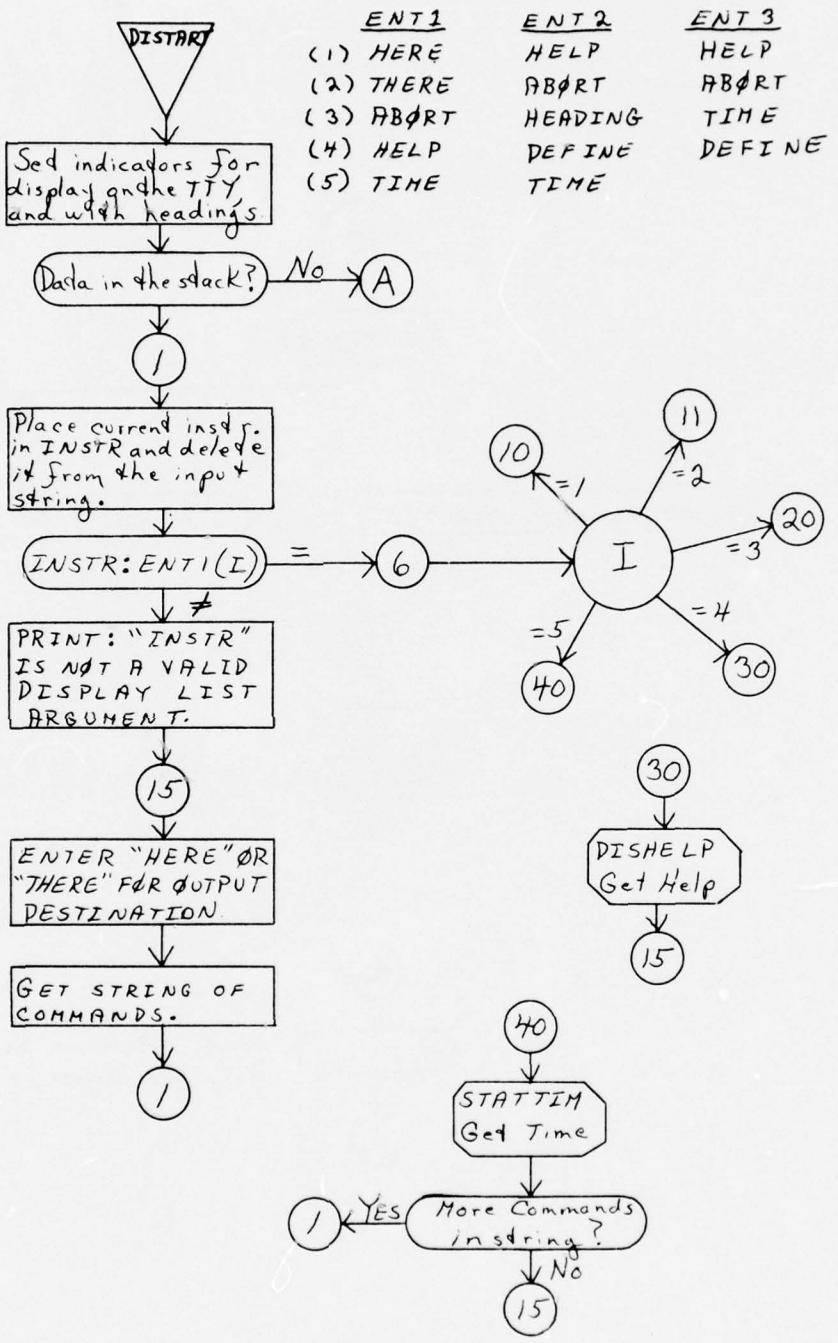

DISPLAY (KB, COUNT)
TAG is the Key Name and is specified in the call to IS subroutine GET. TAG is referenced in the IS FIT, and may take on any value as a Key Name. The IS call to get the referenced data record is CALL GET. The result is to fill the user record area with the retrieved data.
Initialize record counter. PAGE = 0
To determine the destination of the output, whether headings on the output are desired, and the display type or format.
DISTART
TTY or printer output
PRINTER
THERE
TTY
Set line count LINECNT = 50
Compute unique name for the batch output file.
Get record number. NPAGE = SHIFT(I-1, -9) +1
NPAGE : PAGE
=
2
≠
READMS Read data block From File 2.
PAGE = NPAGE
2
Get next SSN (TAG) From TBLA.
TTY or printer output
Printer
Write to 'TAPE3' the SSN
TTY
GET Get user record
1
This will print on the TTY or on the printer the contents of USERREC.
PRINTER (USERREC)
1
Do the following loop COUNT times I = 1, COUNT
RETURN

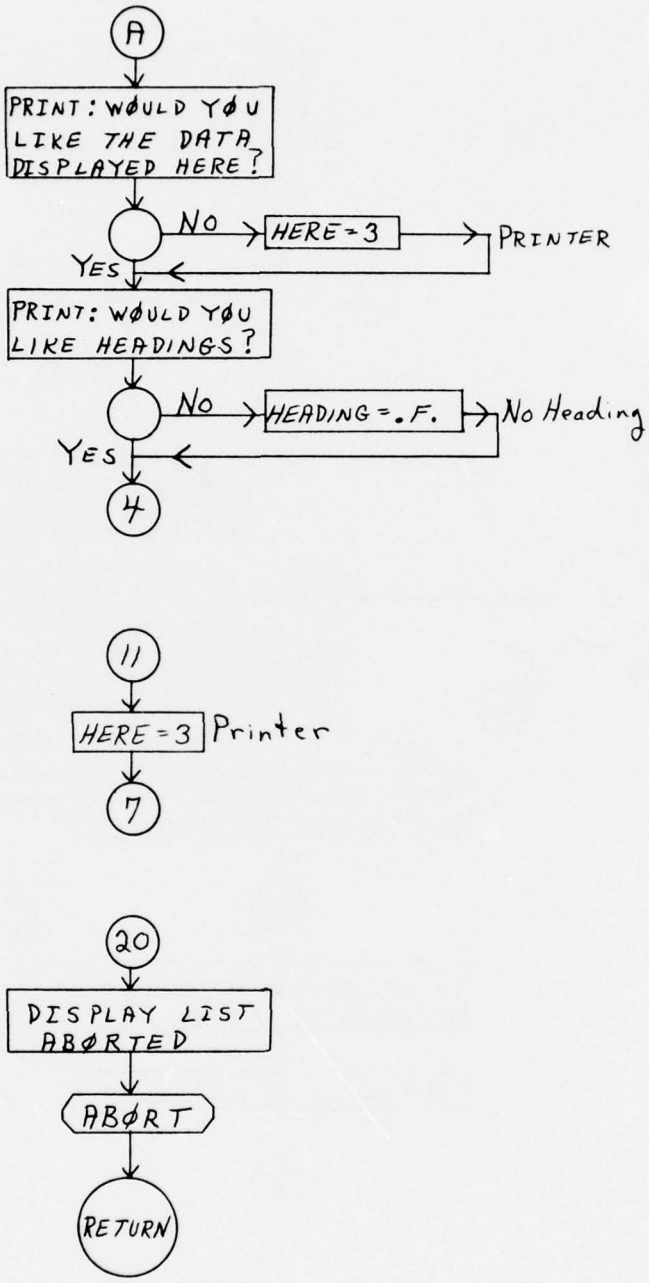

ELCØM (I)
STØP 1
ABØRT 2
TIME 3
HELP 4
ELEMENT (KB,COUNT)
DISTART
Output
Batch
THERE
Set Line Count LINECNT=50
Compute unique name for the batch file.
5
YES
More commands in stack
No
Enter element name...
GETSTRG
Put current element in 'TAG'
Delete the current element from the string.
TAG : ELCØM
=
≠
I
=1 10 STØP
=2 20 ABØRT
=3 TIME 30
=4 40 HELP
FØRGIVE (TAG, 4)
Destination of Output?
Printer
TTY
WRITE "TAG" 'TAPE 3'.
5
GET (IS call)
Read into user area, the array corresponding to the Key (TAG) which has been entered.
ERRCØDE:11
≠
=
PRINT: ELEMENT NAME 'TAG' NØT FØUND. TRY AGAIN.
1
PRINTER (USERREC)
5

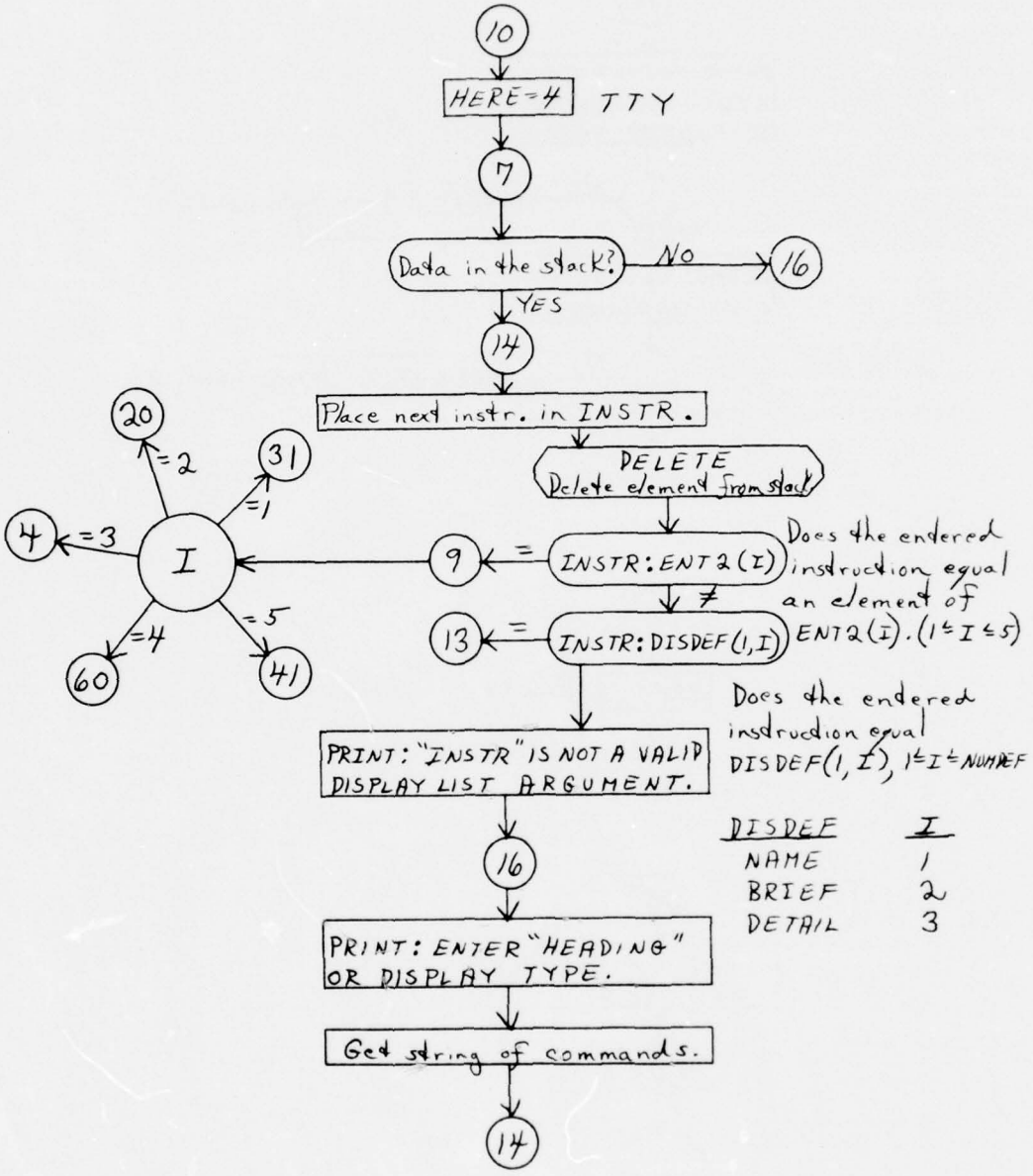

10
RETURN
20
PRINT: ELEMENT ABØRTED.
ABØRT

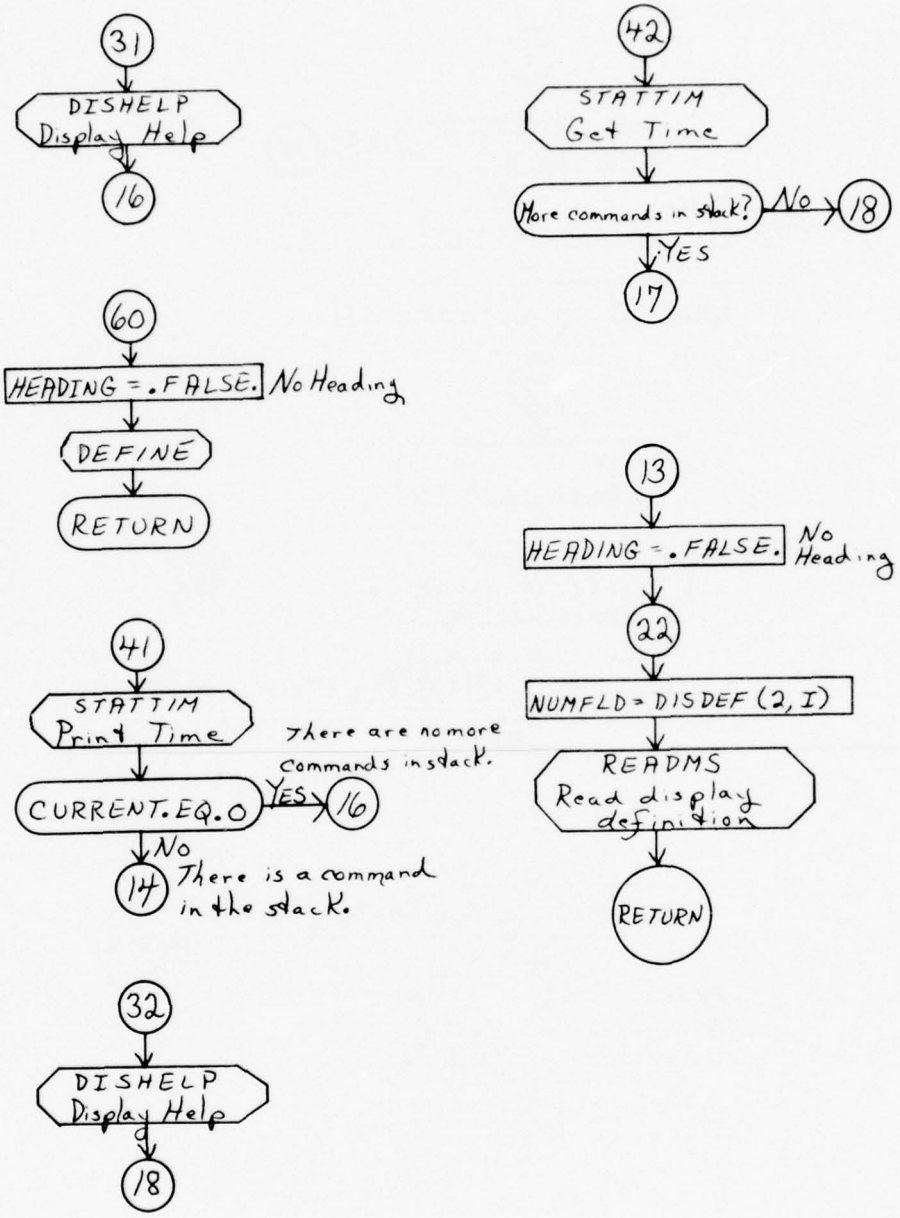

30
STATTIM
5
Print out the time.

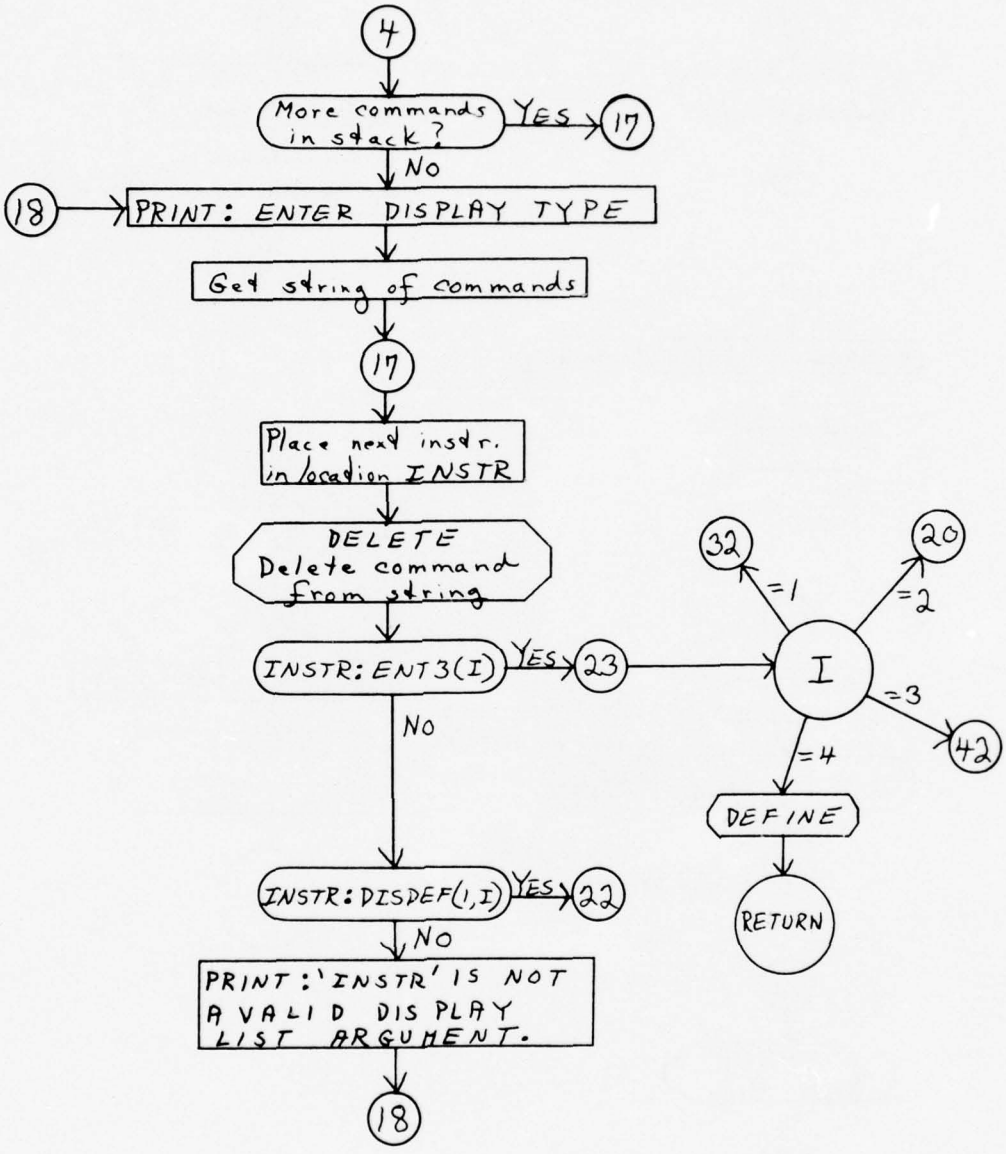

40
ELEHELP
'HELP' Subroutine to give help for element.
1

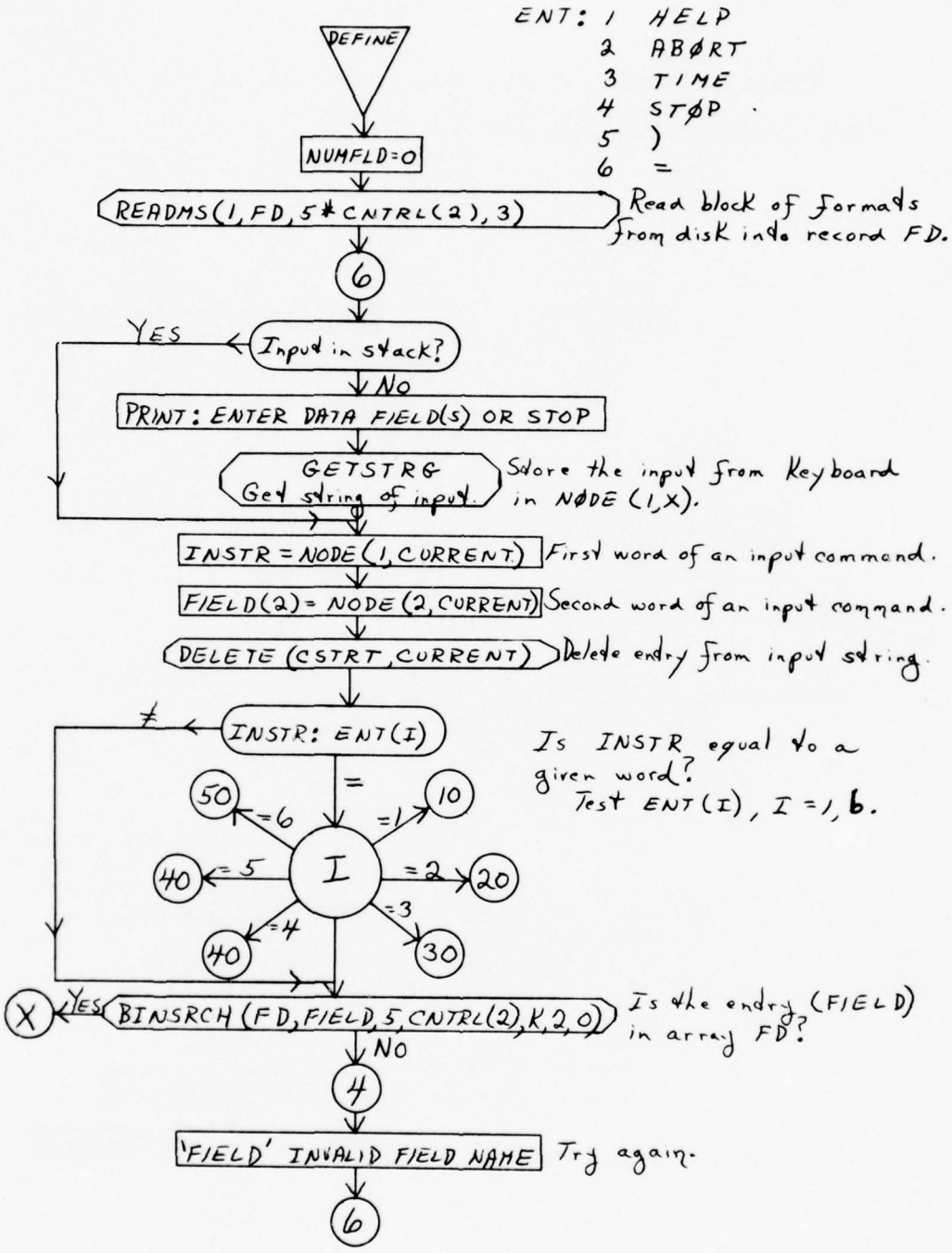

DISTART
Sed indicators for display on the TTY and with headings
Data in the stack?
No
A
1
Place current instr. in INSTR and delete it from the input string.
INSTR: ENT1(I)
=
6
≠
PRINT: "INSTR" IS NOT A VALID DISPLAY LIST ARGUMENT.
15
ENTER "HERE" OR "THERE" FOR OUTPUT DESTINATION
GET STRING OF COMMANDS.
1
ENT1
(1) HERE
(2) THERE
(3) ABORT
(4) HELP
(5) TIME
ENT2
HELP
ABORT
HEADING
DEFINE
TIME
ENT3
HELP
ABORT
TIME
DEFINE
I
10
=1
11
=2
20
=3
30
=4
40
=5
30
DISHELP
Get Help
15
40
STATTIM
Get Time
More Commands in string?
YES
1
No
15

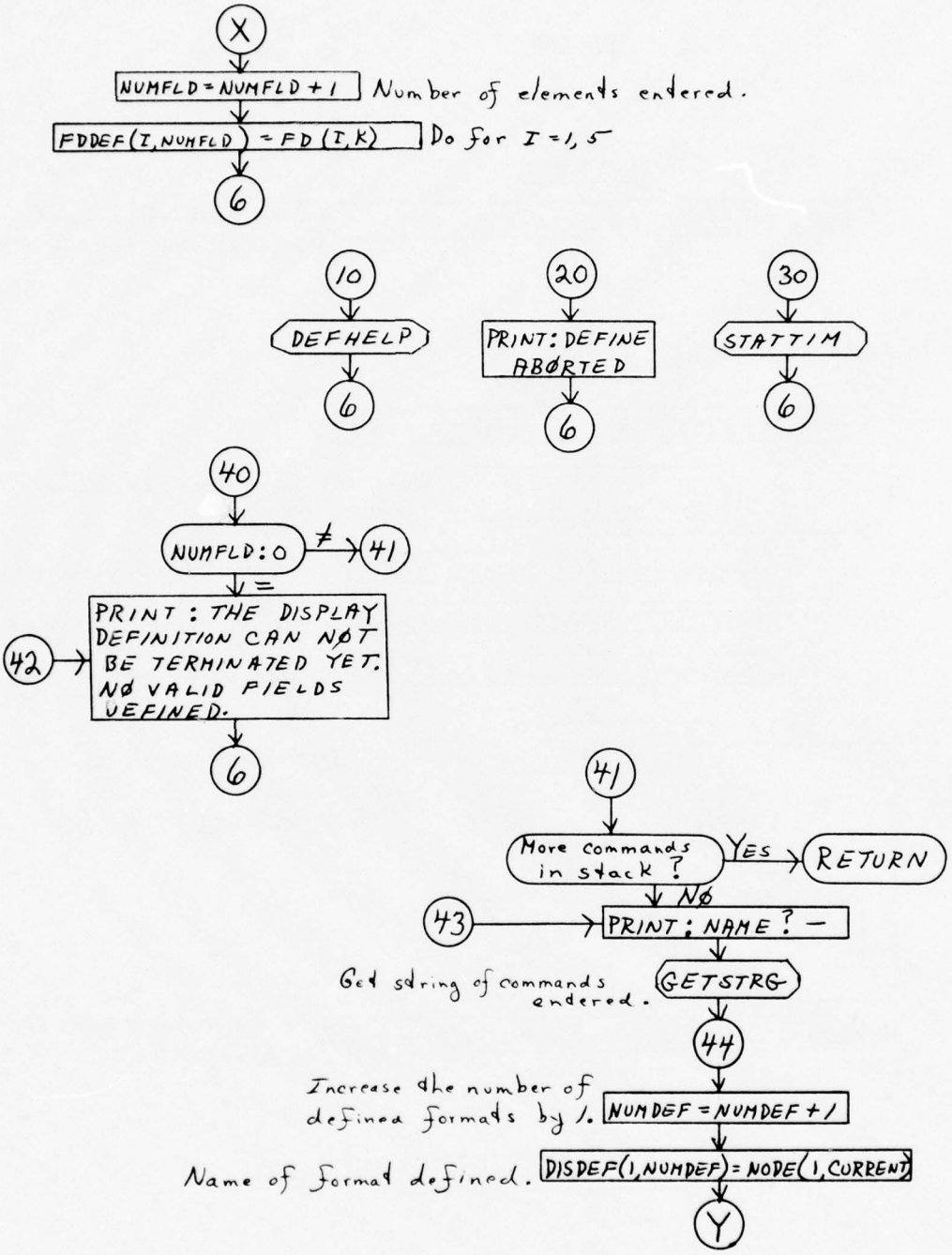

A
PRINT: WØULD YØU LIKE THE DATA DISPLAYED HERE?
No
HERE=3
PRINTER
YES
PRINT: WØULD YØU LIKE HEADINGS?
No
HEADING=.F.
No Heading
YES
4
11
HERE=3
Printer
7
20
DISPLAY LIST ABØRTED
ABØRT
RETURN

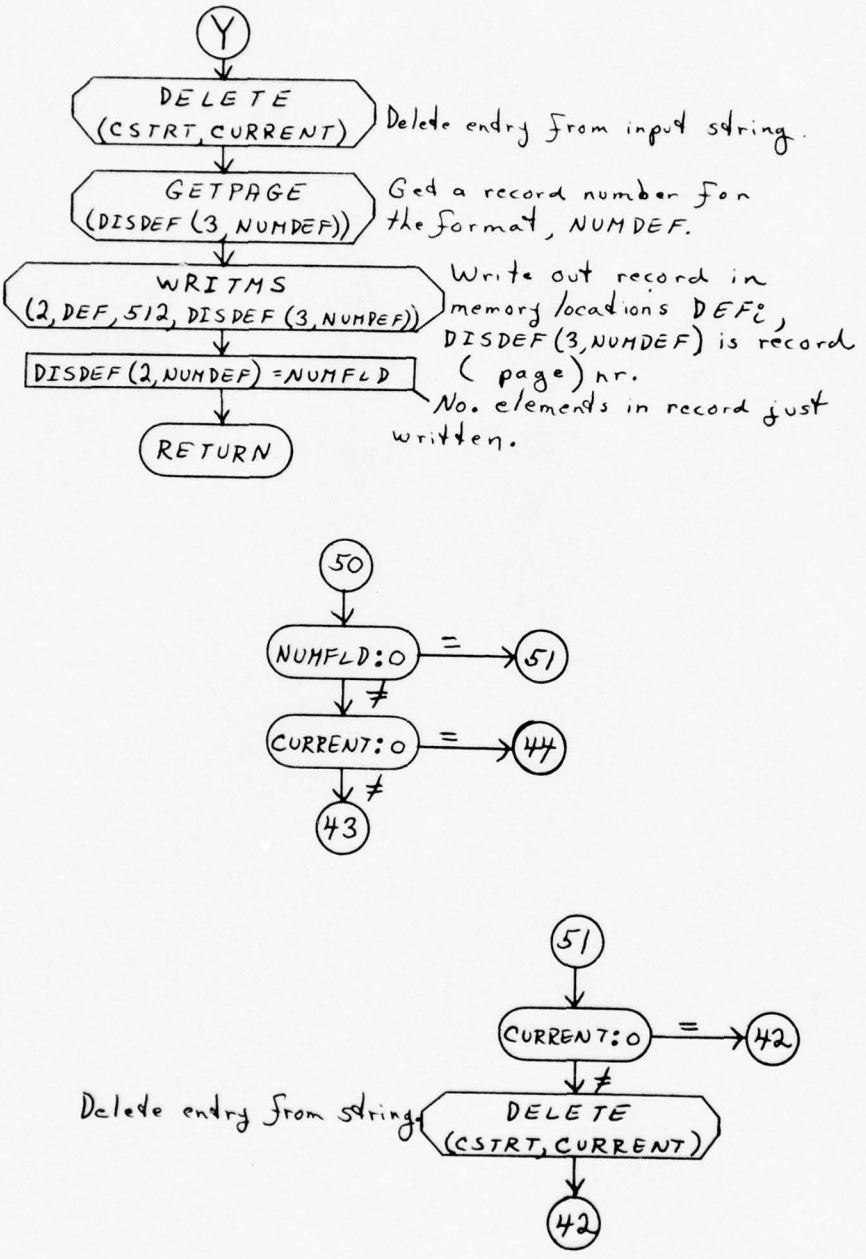

10
HERE=4
TTY
7
Data in the stack?
NO
16
YES
14
Place next instr. in INSTR.
DELETE
Delete element from stack
INSTR:ENT2(I)
Does the entered instruction equal an element of ENT2(I). (1≤I≤5)
9
=
≠
INSTR:DISDEF(1,I)
13
=
Does the entered instruction equal DISDEF(1, I), 1≤I≤NUMDEF
PRINT: "INSTR" IS NOT A VALID DISPLAY LIST ARGUMENT.
16
PRINT: ENTER "HEADING" OR DISPLAY TYPE.
Get string of commands.
14
I
20
=2
31
=1
4
=3
60
=4
41
=5
DISDEF
I
NAME
1
BRIEF
2
DETAIL
3

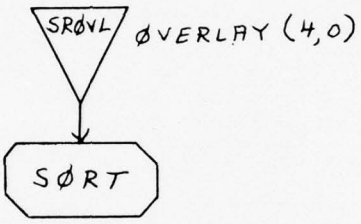

31
DISHELP
Display Help
16
42
STATTIM
Get Time
More commands in stack?
No
18
YES
17
60
HEADING = .FALSE.
No Heading
DEFINE
RETURN
13
HEADING = .FALSE.
No Heading
22
NUMFLD = DISDEF (2, I)
READMS
Read display definition
RETURN
41
STATTIM
Print Time
There are no more commands in stack.
CURRENT.EQ.0
YES
16
No
14
There is a command in the stack.
32
DISHELP
Display Help
18

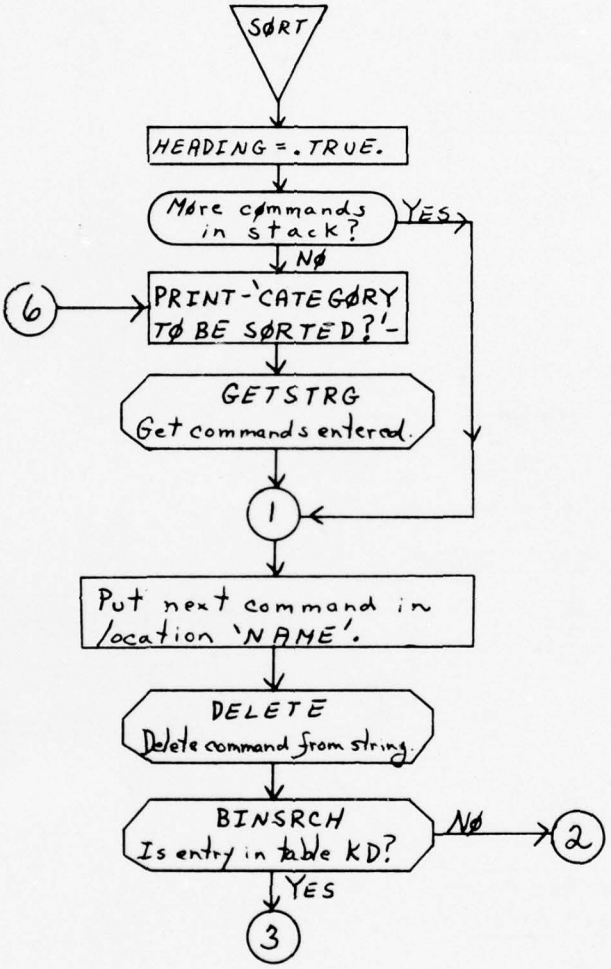

4
More commands in stack?
YES
17
NO
18
PRINT: ENTER DISPLAY TYPE
Get string of commands
17
Place next instr. in location INSTR
DELETE
Delete command from string
INSTR: ENT3(I)
YES
23
NO
I
32
=1
20
=2
=3
42
=4
DEFINE
RETURN
INSTR: DISDEF(1,I)
YES
22
NO
PRINT: 'INSTR' IS NOT A VALID DISPLAY LIST ARGUMENT.
18

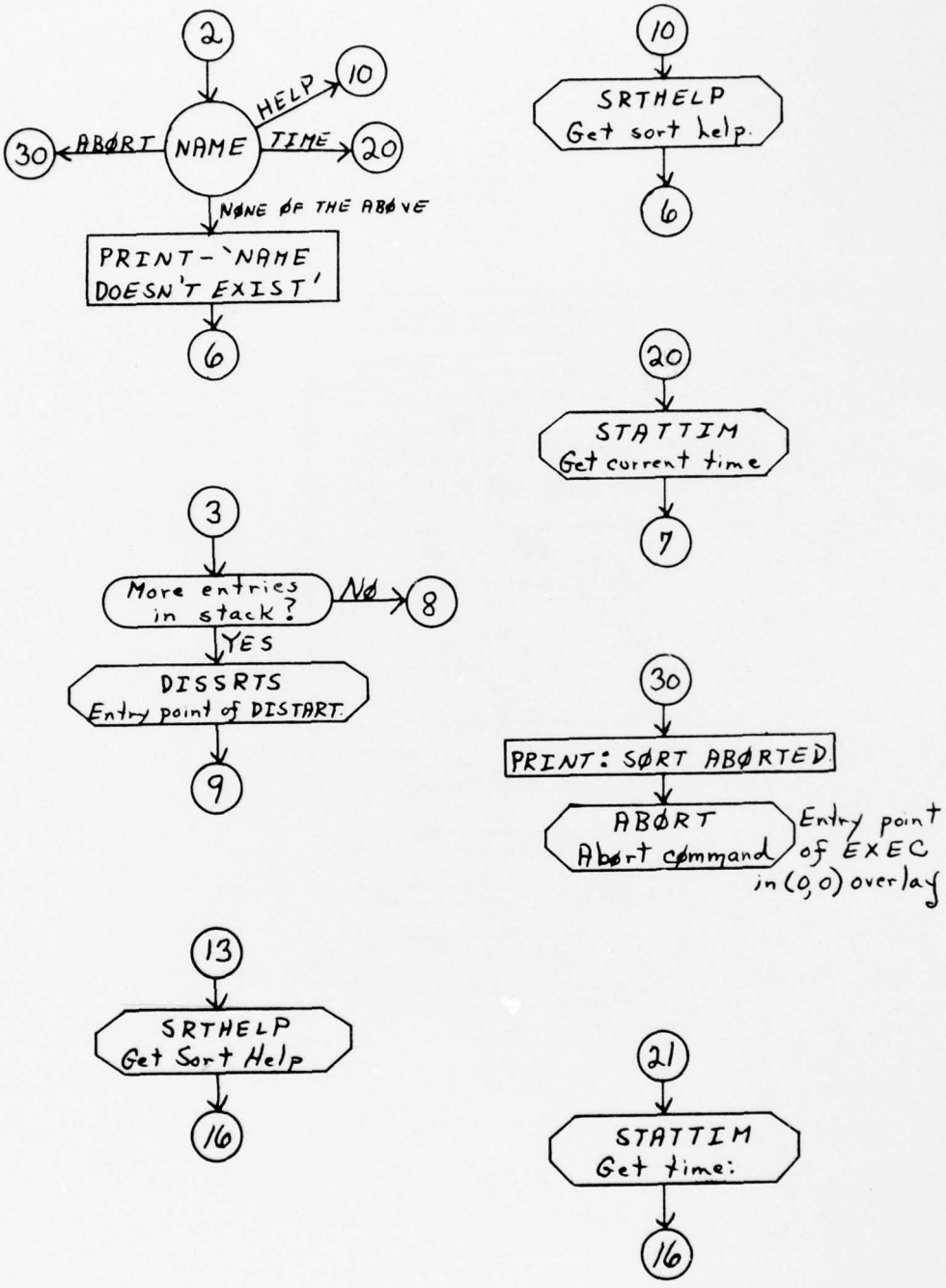

ENT: 1 HELP
2 ABØRT
3 TIME
4 STØP
5)
6 =
DEFINE
NUMFLD=0
READMS(1,FD,5*CNTRL(2),3)
Read block of formats from disk into record FD.
6
YES
Input in stack?
No
PRINT: ENTER DATA FIELD(S) OR STOP
GETSTRG
Get string of input.
Store the input from Keyboard in NØDE(1,X).
INSTR=NODE(1,CURRENT)
First word of an input command.
FIELD(2)=NODE(2,CURRENT)
Second word of an input command.
DELETE(CSTRT,CURRENT)
Delete entry from input string.
≠
INSTR: ENT(I)
Is INSTR equal to a given word?
Test ENT(I), I=1,6.
=
50
=6
10
=1
40
=5
I
=2
20
=3
=4
40
30
X
YES
BINSRCH(FD,FIELD,5,CNTRL(2),K,2,0)
Is the entry (FIELD) in array FD?
No
4
'FIELD' INVALID FIELD NAME
Try again.
6

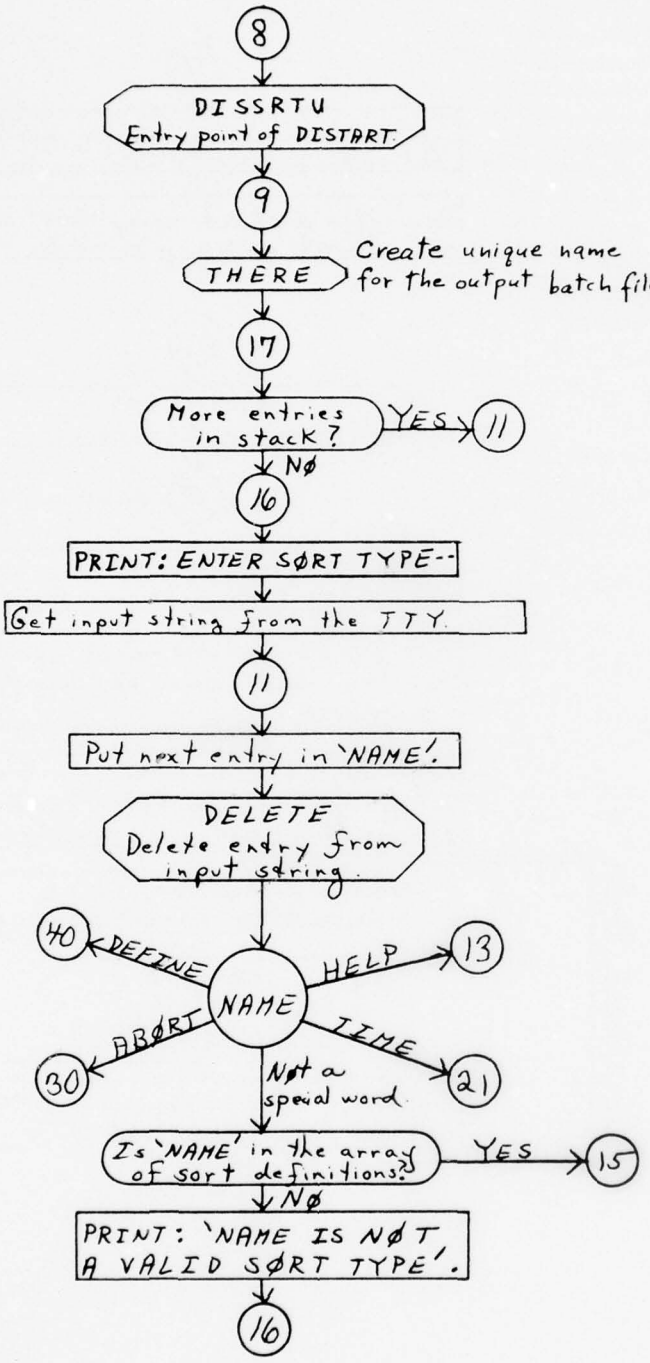

X
NUMFLD = NUMFLD + 1
Number of elements entered.
FDDEF(I, NUMFLD) = FD(I, K)
Do for I = 1, 5
6
10
DEFHELP
6
20
PRINT: DEFINE ABORTED
6
30
STATTIM
6
40
NUMFLD: 0
≠
41
=
42
PRINT: THE DISPLAY DEFINITION CAN NOT BE TERMINATED YET. NO VALID FIELDS DEFINED.
6
41
More commands in stack ?
YES
RETURN
NO
43
PRINT: NAME? —
Get string of commands entered.
GETSTRG
44
Increase the number of defined formats by 1.
NUMDEF = NUMDEF + 1
Name of format defined.
DISDEF(1, NUMDEF) = NODE(1, CURRENT)
Y

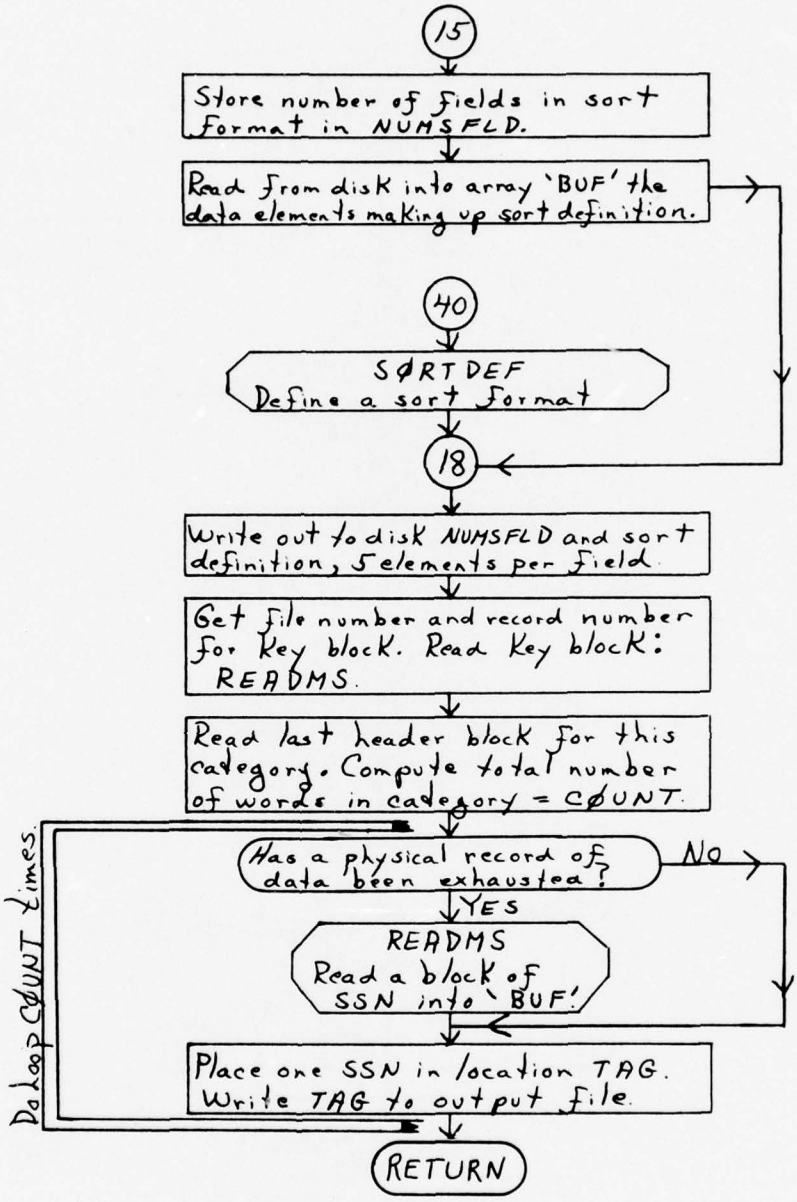

Y
DELETE (CSTRT, CURRENT)
Delete entry from input string.
GETPAGE (DISDEF (3, NUMDEF))
Get a record number for the format, NUMDEF.
WRITMS (2, DEF, 512, DISDEF (3, NUMPEF))
Write out record in memory locations DEFi, DISDEF (3, NUMDEF) is record (page) nr.
DISDEF (2, NUMDEF) = NUMFLD
No. elements in record just written.
RETURN
50
NUMFLD:0
=
51
≠
CURRENT:0
=
44
≠
43
51
CURRENT:0
=
42
≠
Delete entry from string
DELETE (CSTRT, CURRENT)
42

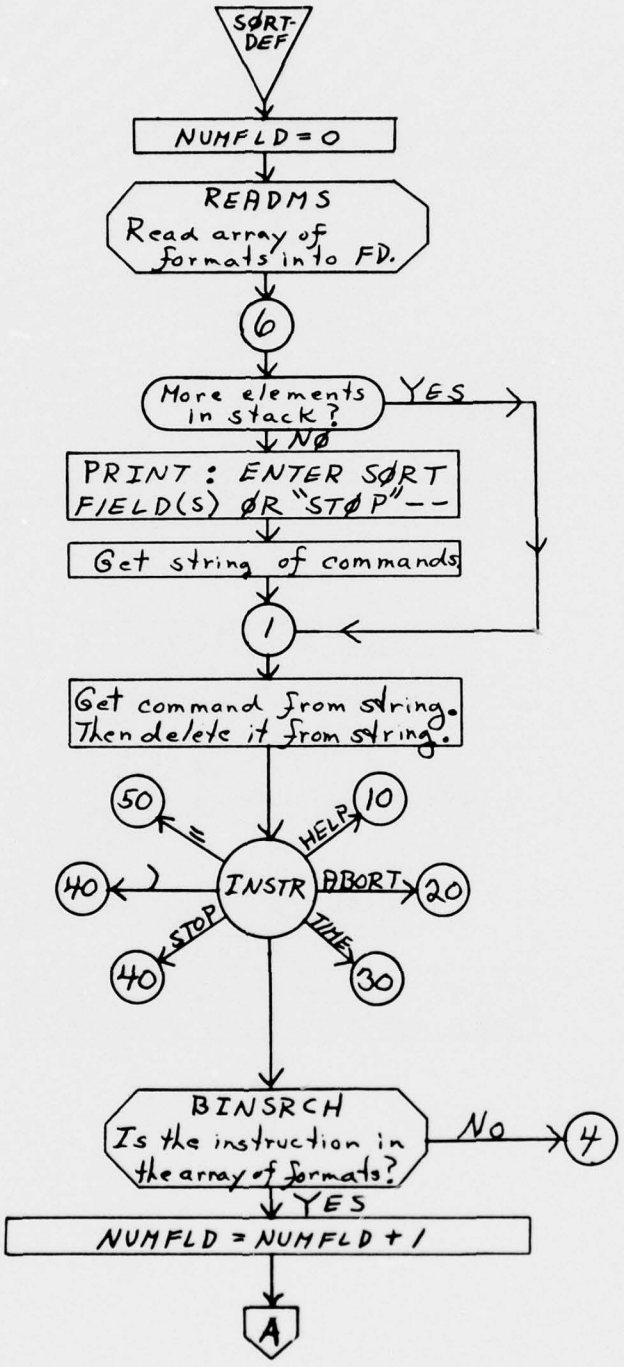

SRØVL
ØVERLAY (4,0)
SØRT

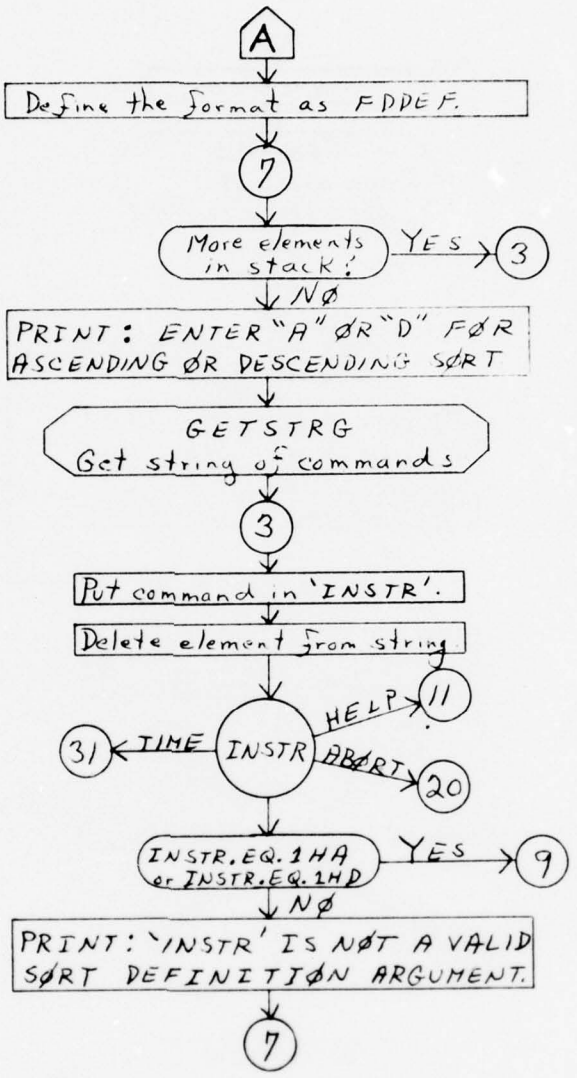

SØRT
HEADING = .TRUE.
More commands in stack?
YES
NØ
6
PRINT-'CATEGØRY TØ BE SØRTED?'-
GETSTRG
Get commands entered.
1
Put next command in location 'NAME'.
DELETE
Delete command from string.
BINSRCH
Is entry in table KD?
NØ
2
YES
3

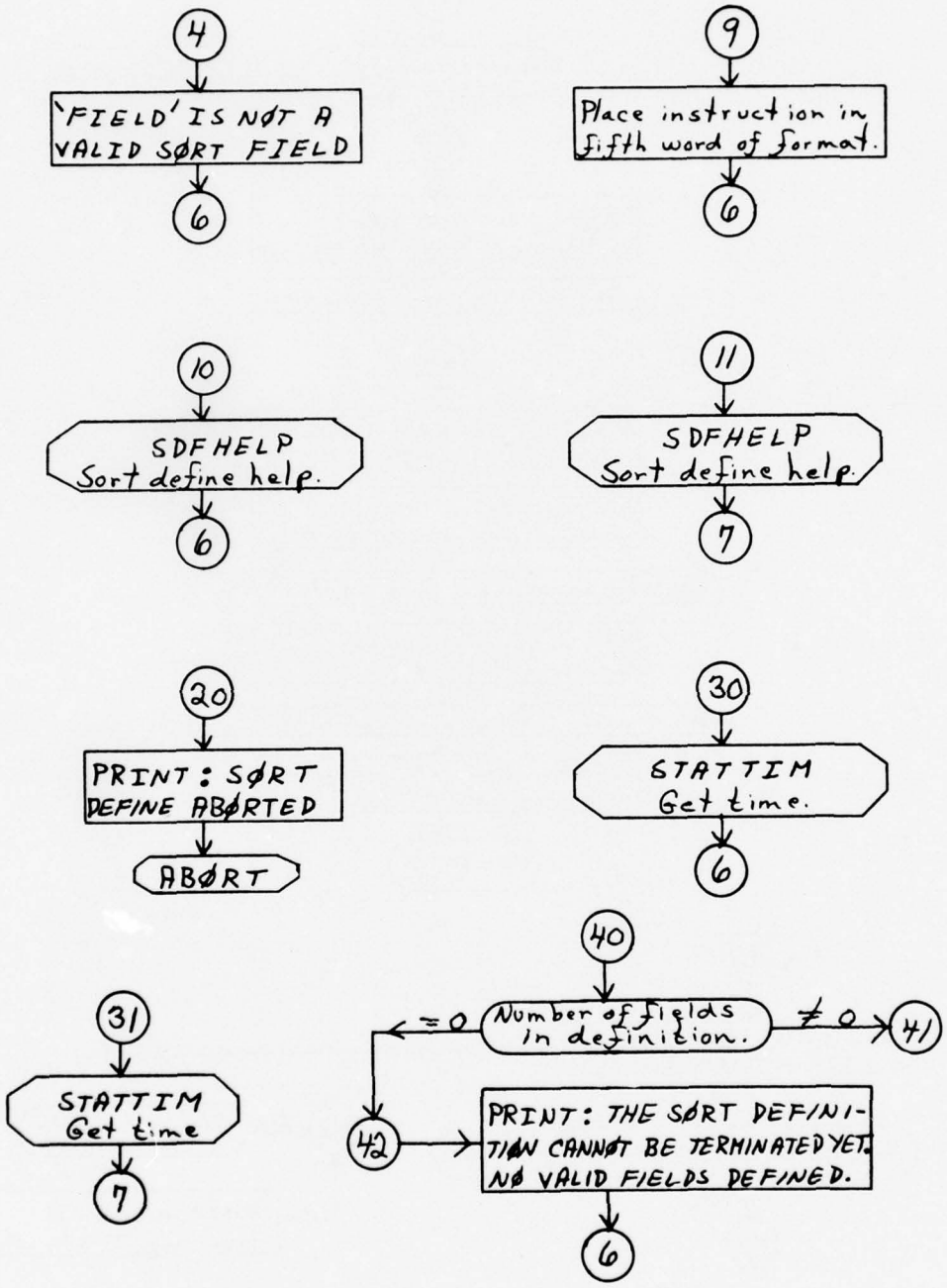

2
NAME
HELP
10
TIME
20
ABØRT
30
NØNE ØF THE ABØVE
PRINT-`NAME DOESN'T EXIST'
6
10
SRTHELP
Get sort help.
6
20
STATTIM
Get current time
7
3
More entries in stack?
NØ
8
YES
DISSRTS
Entry point of DISTART.
9
30
PRINT: SØRT ABØRTED
ABØRT
Abort command
Entry point of EXEC in (0,0) overlay
13
SRTHELP
Get Sort Help
16
21
STATTIM
Get time:
16

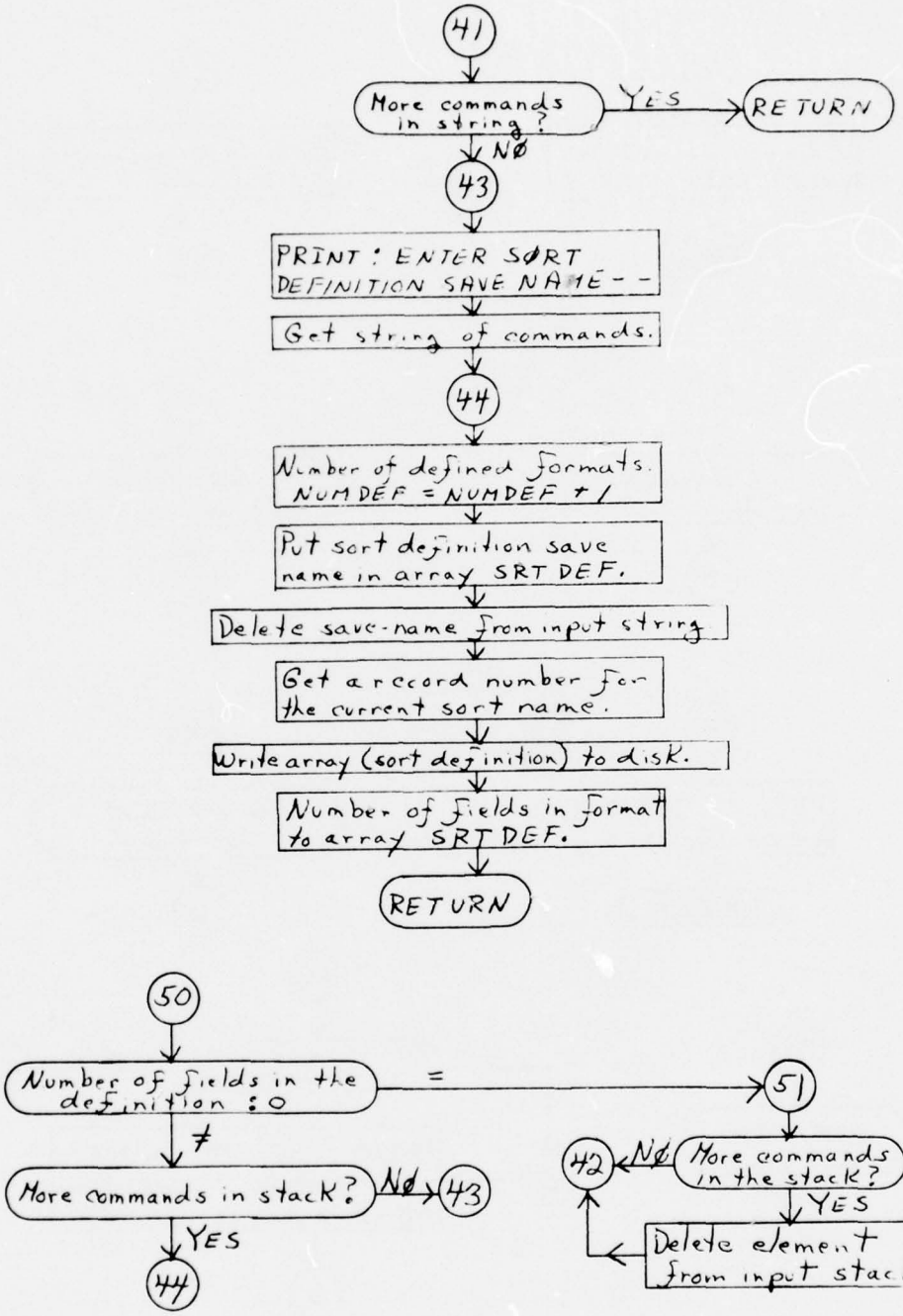

8
DISSRTU
Entry point of DISTART
9
THERE
Create unique name for the output batch file.
17
More entries in stack?
YES
11
NØ
16
PRINT: ENTER SØRT TYPE--
Get input string from the TTY.
11
Put next entry in 'NAME'.
DELETE
Delete entry from input string.
40
DEFINE
NAME
HELP
13
ABØRT
30
TIME
21
Not a special word.
Is 'NAME' in the array of sort definitions?
YES
15
NØ
PRINT: 'NAME IS NØT A VALID SØRT TYPE'.
16

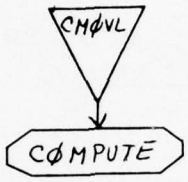

15
Store number of fields in sort format in NUMSFLD.
Read from disk into array 'BUF' the data elements making up sort definition.
40
SORTDEF
Define a sort format
18
Write out to disk NUMSFLD and sort definition, 5 elements per field.
Get file number and record number for key block. Read key block: READMS.
Read last header block for this category. Compute total number of words in category = COUNT.
Do loop COUNT times.
Has a physical record of data been exhausted?
NO
YES
READMS
Read a block of SSN into 'BUF'
Place one SSN in location TAG. Write TAG to output file.
RETURN


SØRT-DEF
NUMFLD = 0
READMS
Read array of formats into FD.
6
More elements in stack?
YES
NØ
PRINT : ENTER SØRT FIELD(S) ØR "STØP"--
Get string of commands
1
Get command from string. Then delete it from string.
50
=
HELP
10
40
)
INSTR
ABORT
20
STOP
40
TIME
30
BINSRCH
Is the instruction in the array of formats?
No
4
YES
NUMFLD = NUMFLD + 1
A


A
Define the format as FDDEF.
7
More elements in stack?
YES
3
NØ
PRINT: ENTER "A" ØR "D" FØR ASCENDING ØR DESCENDING SØRT
GETSTRG
Get string of commands
3
Put command in 'INSTR'.
Delete element from string
INSTR
HELP
11
31
TIME
ABØRT
20
INSTR.EQ.1HA or INSTR.EQ.1HD
YES
9
NØ
PRINT: 'INSTR' IS NØT A VALID SØRT DEFINITIØN ARGUMENT.
7

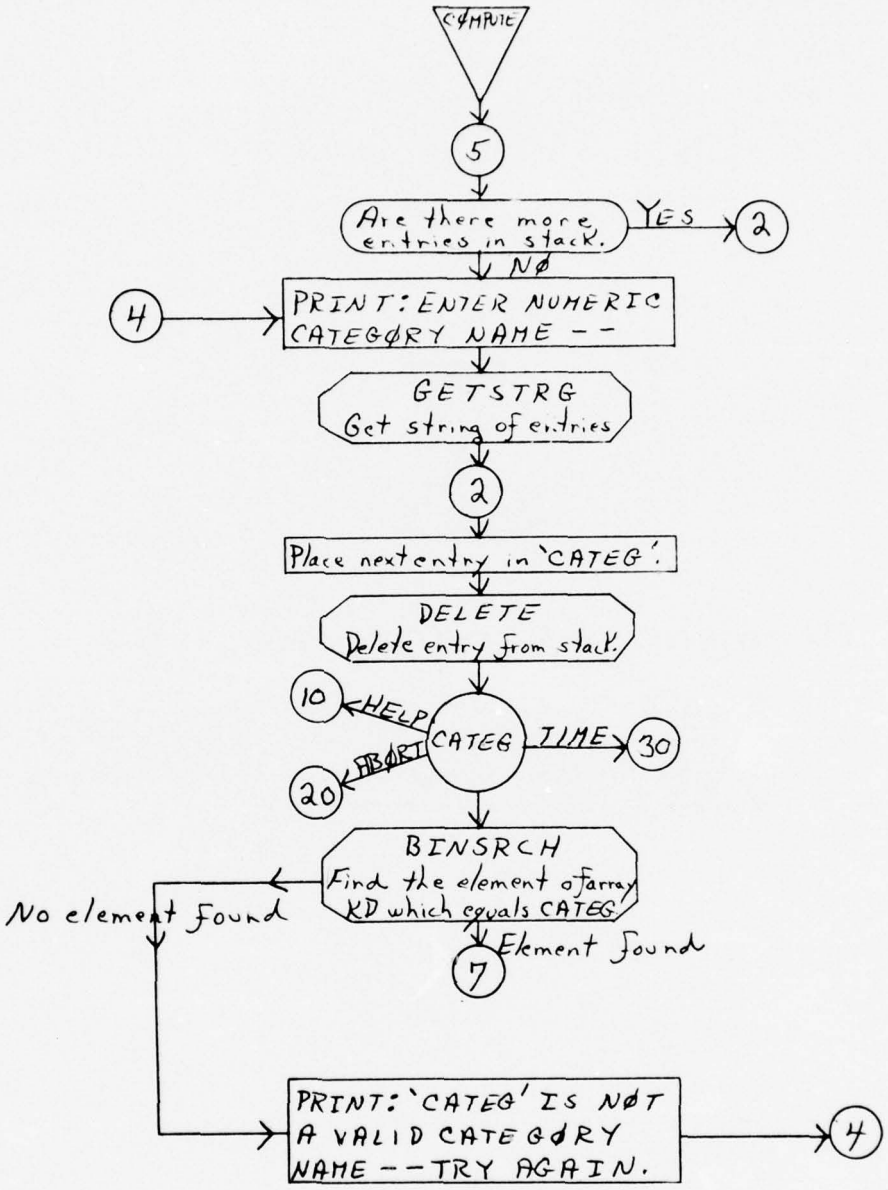

4
'FIELD' IS NØT A VALID SØRT FIELD
6
9
Place instruction in fifth word of format.
6
10
SDFHELP
Sort define help.
6
11
SDFHELP
Sort define help.
7
20
PRINT: SØRT DEFINE ABØRTED
ABØRT
30
STATTIM
Get time.
6
31
STATTIM
Get time
7
40
Number of fields in definition.
= 0
≠ 0
41
42
PRINT: THE SØRT DEFINITION CANNØT BE TERMINATED YET. NØ VALID FIELDS DEFINED.
6

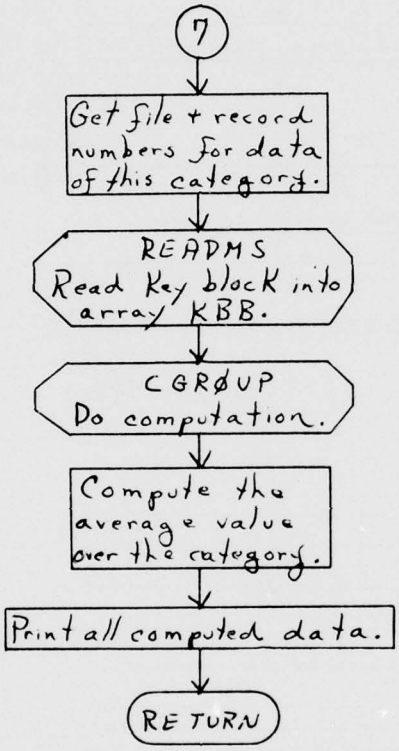

41
More commands in string?
YES
RETURN
NØ
43
PRINT : ENTER SORT DEFINITION SAVE NAME - -
Get string of commands.
44
Number of defined formats. NUMDEF = NUMDEF + 1
Put sort definition save name in array SRT DEF.
Delete save-name from input string
Get a record number for the current sort name.
Write array (sort definition) to disk.
Number of fields in format to array SRT DEF.
RETURN
50
Number of fields in the definition : 0
=
51
≠
More commands in stack?
NØ
43
YES
44
42
NØ
More commands in the stack?
YES
Delete element from input stack.

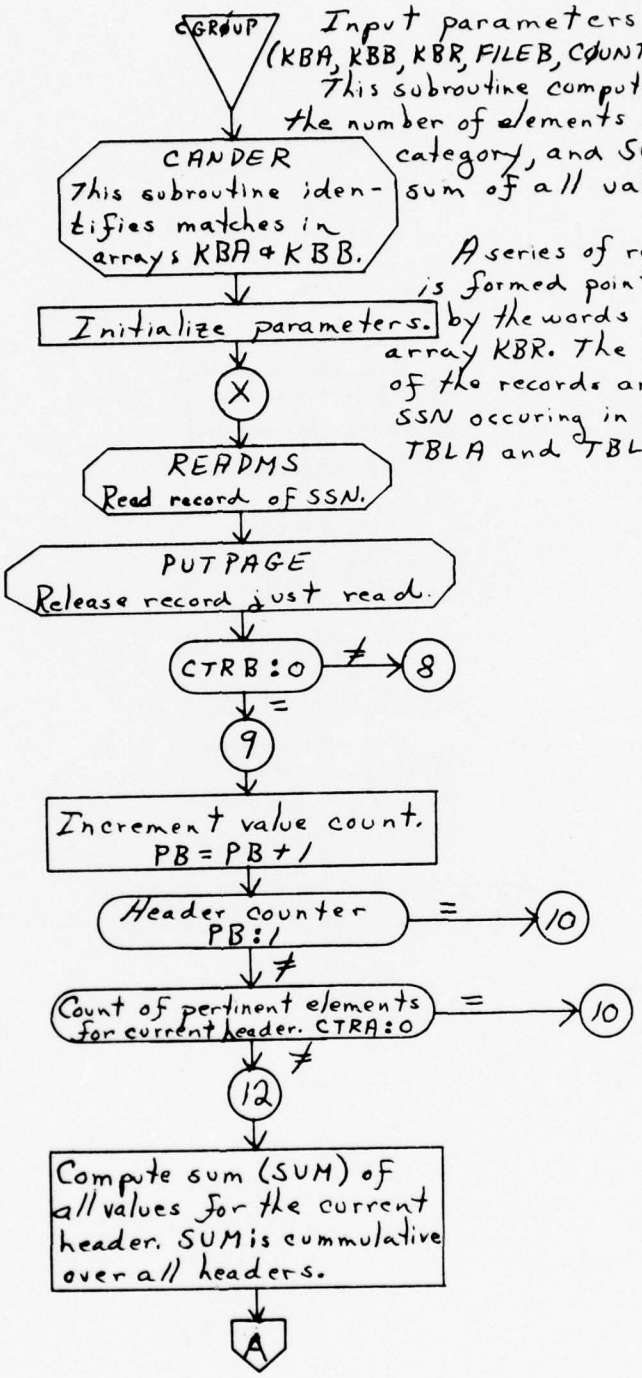


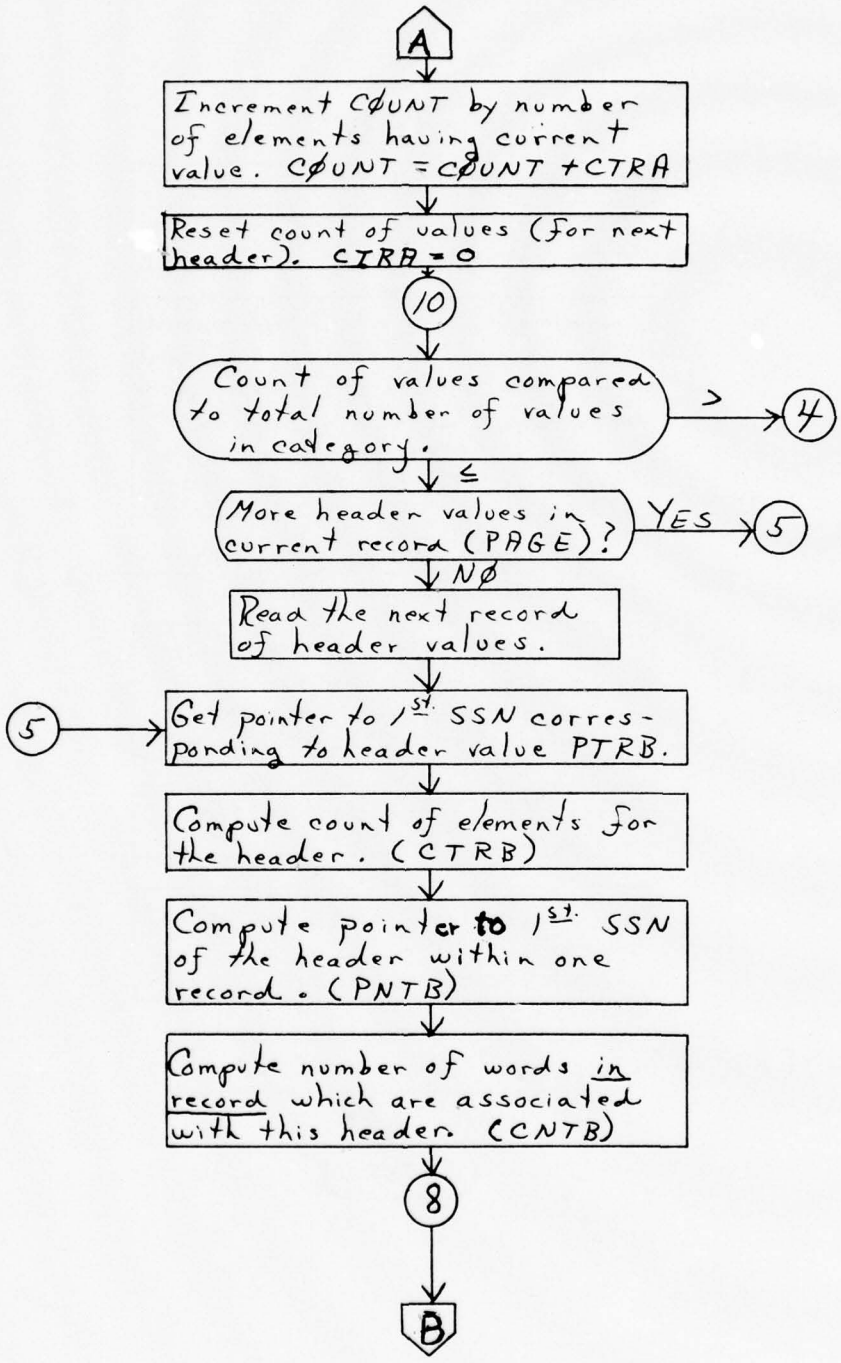
The purpose of ØVERLAY(5,0), CMØVL, is to:

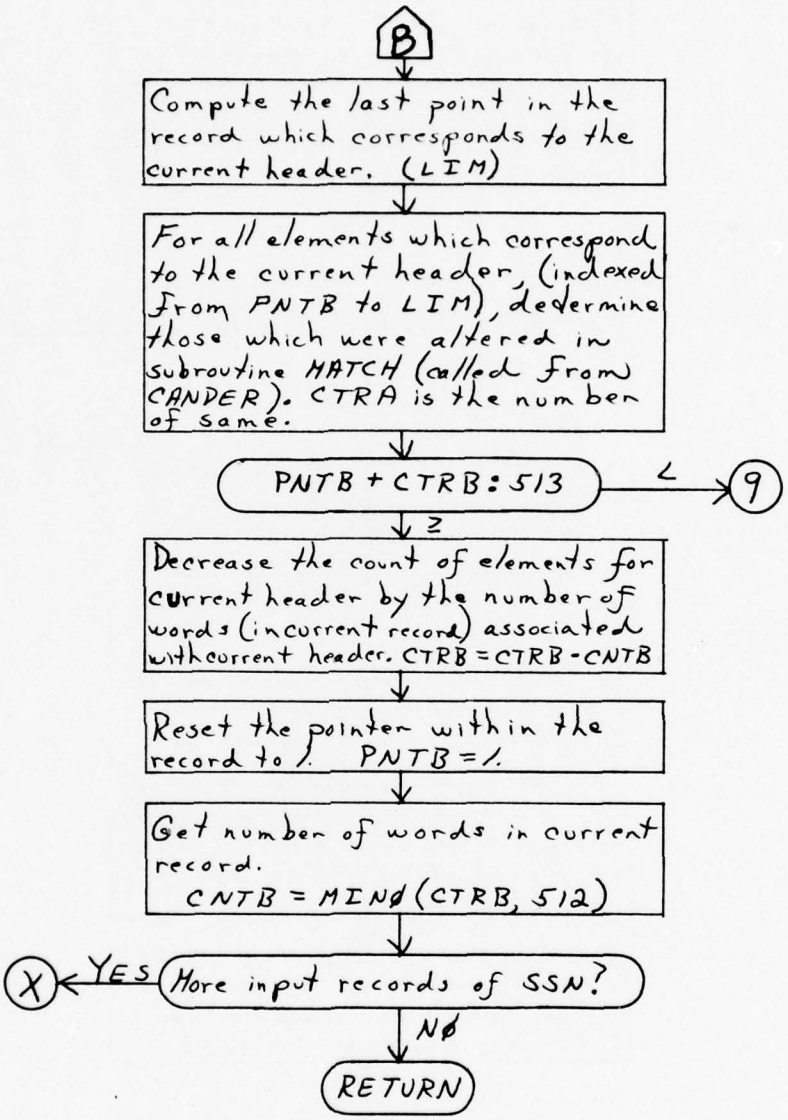
1. Compute the number of elements in a category.
2. Compute the sum of the values of the elements.
3. Compute the average value over the category.
4. Print out this information.

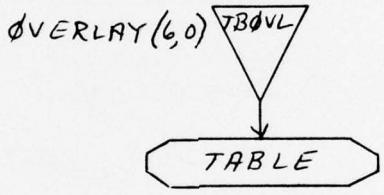

COMPUTE
5
Are there more entries in stack.
YES
2
NØ
4
PRINT: ENTER NUMERIC CATEGØRY NAME --
GETSTRG
Get string of entries
2
Place next entry in 'CATEG'.
DELETE
Delete entry from stack.
10
HELP
CATEG
TIME
30
ABØRT
20
BINSRCH
Find the element of array KD which equals CATEG
No element Found
Element found
7
PRINT: 'CATEG' IS NØT A VALID CATEGØRY NAME -- TRY AGAIN.
4


7
Get file + record numbers for data of this category.
READMS
Read Key block into array KBB.
CGRØUP
Do computation.
Compute the average value over the category.
Print all computed data.
RETURN


GROUP
Input parameters: (KBA, KBB, KBR, FILEB, COUNT, SUM) This subroutine computes COUNT, the number of elements in the category, and SUM, the sum of all values.
CANDER
This subroutine identifies matches in arrays KBA & KBB.
A series of records is formed pointed to by the words in array KBR. The elements of the records are those SSN occuring in both TBLA and TBLB.
Initialize parameters.
X
READMS
Read record of SSN.
PUTPAGE
Release record just read.
CTRB:0
≠
8
=
9
Increment value count.
PB = PB + 1
Header counter
PB:1
=
10
≠
Count of pertinent elements for current header. CTRA:0
=
10
≠
12
Compute sum (SUM) of all values for the current header. SUM is cummulative over all headers.
A


A
Increment COUNT by number of elements having current value. COUNT = COUNT + CTRA
Reset count of values (for next header). CTRA = 0
10
Count of values compared to total number of values in category.
>
4
≤
More header values in current record (PAGE)?
YES
5
NO
Read the next record of header values.
5
Get pointer to 1st SSN corresponding to header value PTRB.
Compute count of elements for the header. (CTRB)
Compute pointer to 1st SSN of the header within one record. (PNTB)
Compute number of words in record which are associated with this header. (CNTB)
8
B


B
Compute the last point in the record which corresponds to the current header. (LIM)
For all elements which correspond to the current header, (indexed from PNTB to LIM), determine those which were altered in subroutine MATCH (called from CANDER). CTRA is the number of same.
PNTB + CTRB : 513
<
9
≥
Decrease the count of elements for current header by the number of words (in current record) associated with current header. CTRB = CTRB - CNTB
Reset the pointer within the record to 1. PNTB = 1.
Get number of words in current record.
CNTB = MIN0 (CTRB, 512)
X
YES
More input records of SSN?
N0
RETURN


ØVERLAY(6,0)
TBØVL
TABLE

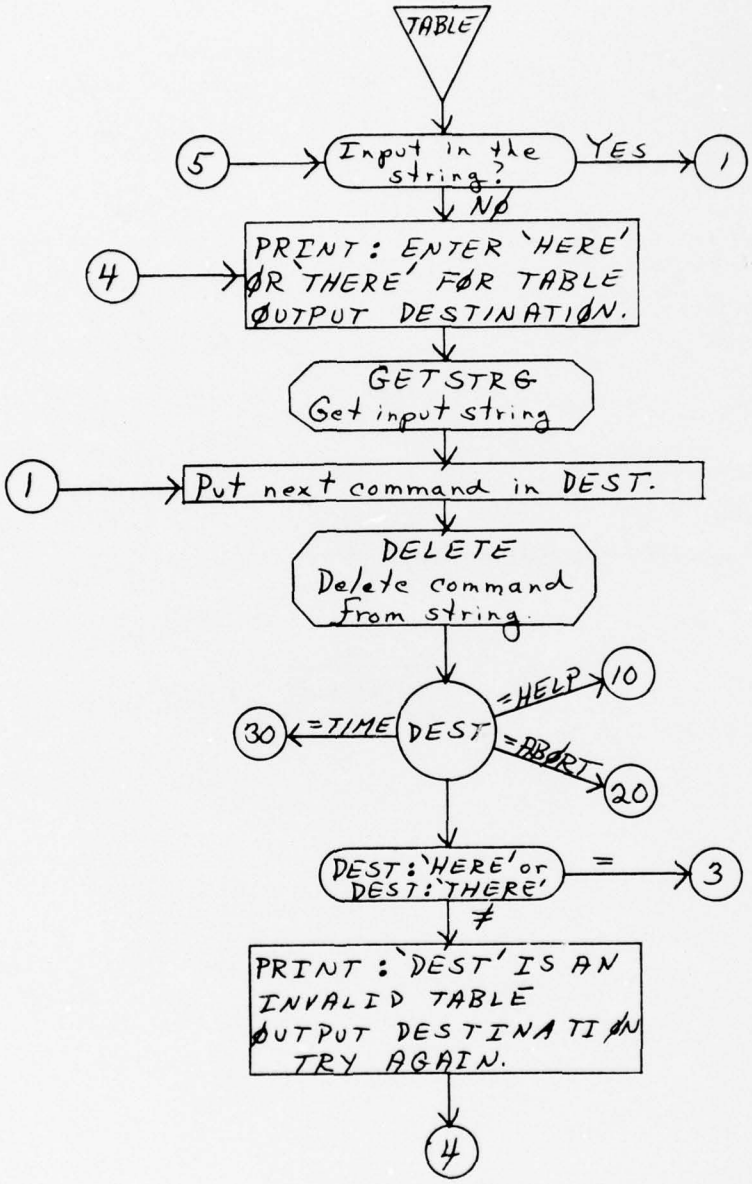
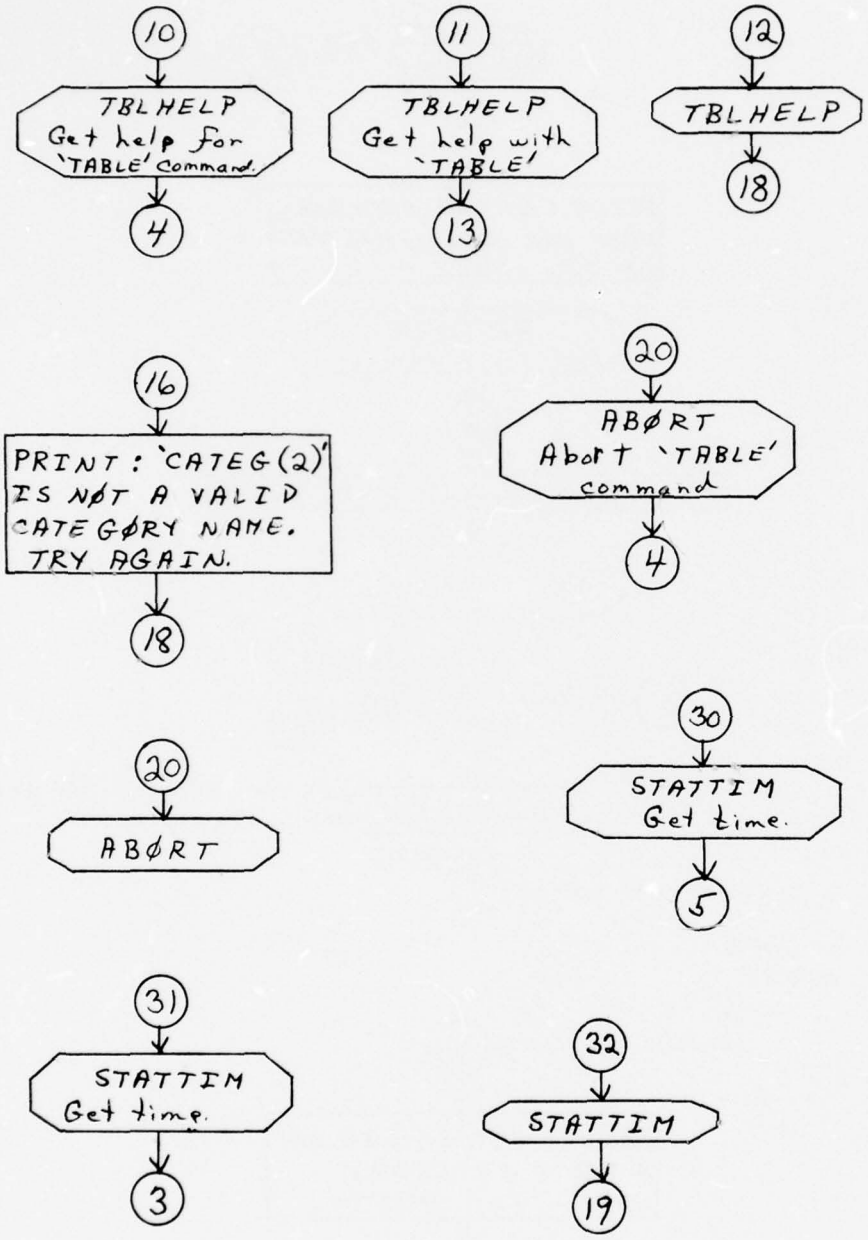
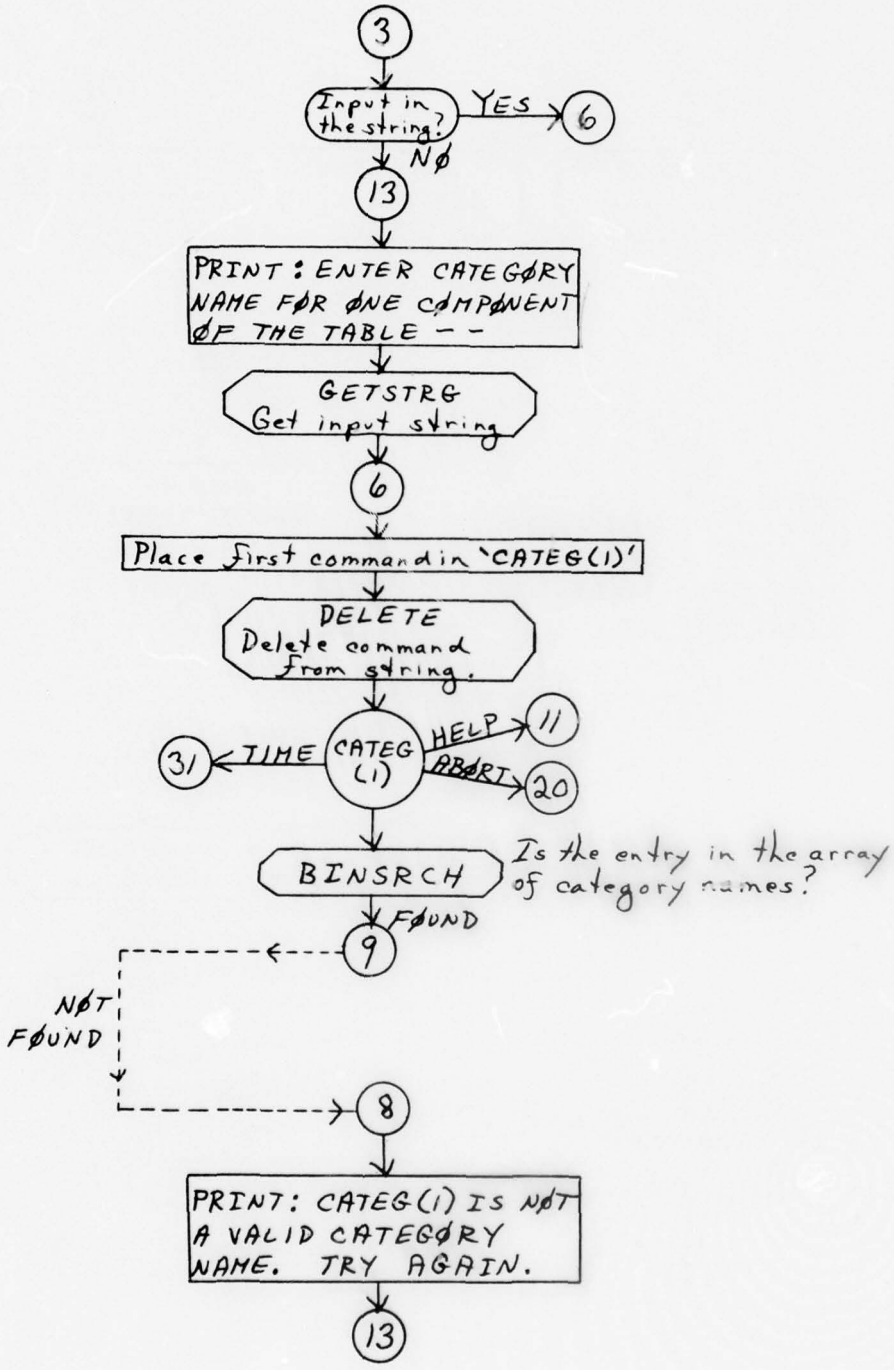
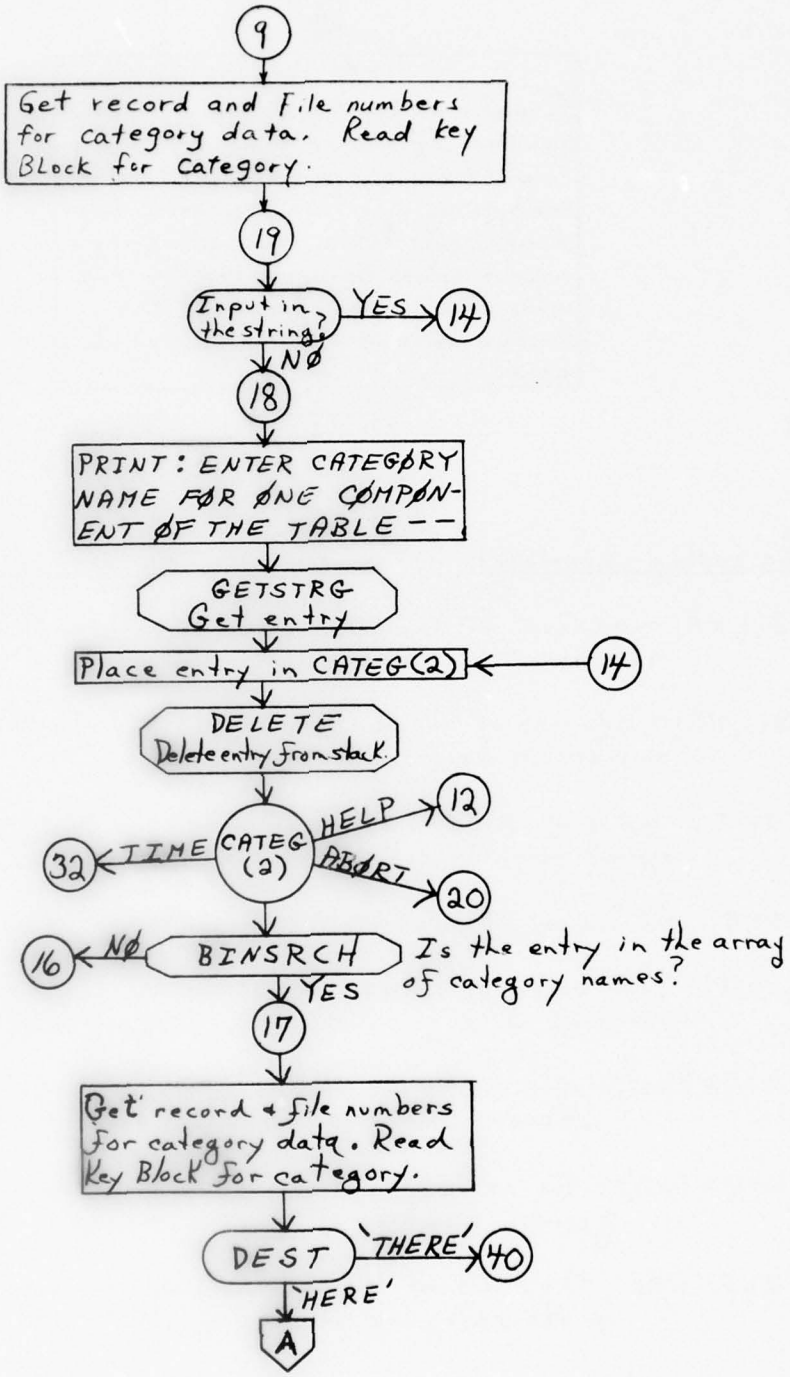
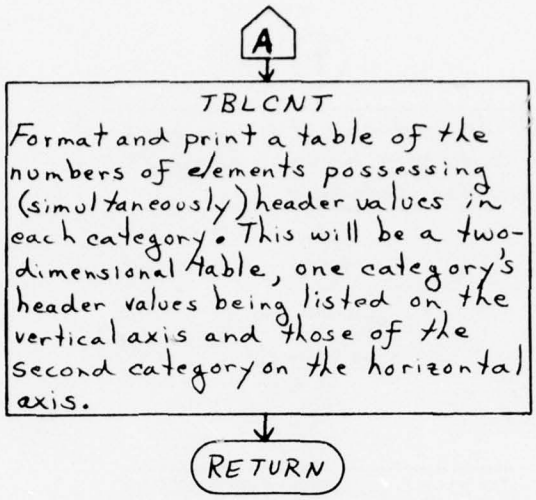


TABLE
5
Input in the string?
YES
1
NO
4
PRINT: ENTER 'HERE' ØR 'THERE' FØR TABLE ØUTPUT DESTINATIØN.
GETSTRG
Get input string
1
Put next command in DEST.
DELETE
Delete command from string.
DEST
=HELP
10
30
=TIME
=ABØRT
20
DEST:'HERE' or DEST:'THERE'
=
3
≠
PRINT: 'DEST' IS AN INVALID TABLE ØUTPUT DESTINATIØN TRY AGAIN.
4


10
TBLHELP
Get help for
'TABLE' command.
4
11
TBLHELP
Get help with
'TABLE'
13
12
TBLHELP
18
16
PRINT: 'CATEG(2)'
IS NØT A VALID
CATEGØRY NAME.
TRY AGAIN.
18
20
ABØRT
Abort 'TABLE'
command
4
20
ABØRT
30
STATTIM
Get time.
5
31
STATTIM
Get time.
3
32
STATTIM
19


3
Input in the string?
YES
6
NØ
13
PRINT: ENTER CATEGØRY NAME FØR ØNE CØMPØNENT ØF THE TABLE --
GETSTRG
Get input string
6
Place first command in 'CATEG(1)'
DELETE
Delete command from string.
CATEG (1)
HELP
11
TIME
31
ABØRT
20
BINSRCH
Is the entry in the array of category names?
FØUND
9
NØT FØUND
8
PRINT: CATEG(1) IS NØT A VALID CATEGØRY NAME. TRY AGAIN.
13


9
Get record and File numbers for category data. Read key Block for Category.
19
Input in the string?
YES
14
NØ
18
PRINT: ENTER CATEGØRY NAME FØR ØNE CØMPØN-ENT ØF THE TABLE --
GETSTRG Get entry
Place entry in CATEG(2)
14
DELETE Delete entry from stack.
CATEG (2)
HELP
12
TIME
32
ABØRT
20
BINSRCH
Is the entry in the array of category names?
NØ
16
YES
17
Get record & file numbers for category data. Read Key Block for category.
DEST
'THERE'
40
'HERE'
A



The calling parameters of subroutine TBLCNT are:

KB(1, NA) - address of Key block of category with the smaller number of header values.

KB(1, NB) - address of Key block of category with the greater number of header values.

LØWA - index of first header of category KB(1, NA) to appear over the columns on a given page.

NUMA - the number of columns on a given page

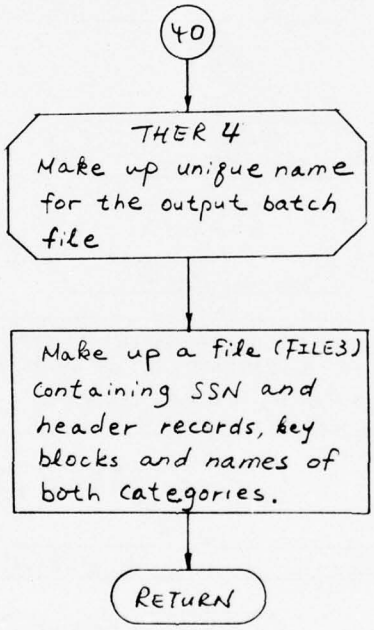
LIMB - the number of header values listed in the vertical direction on a given page. (Category KB(1, NB)).

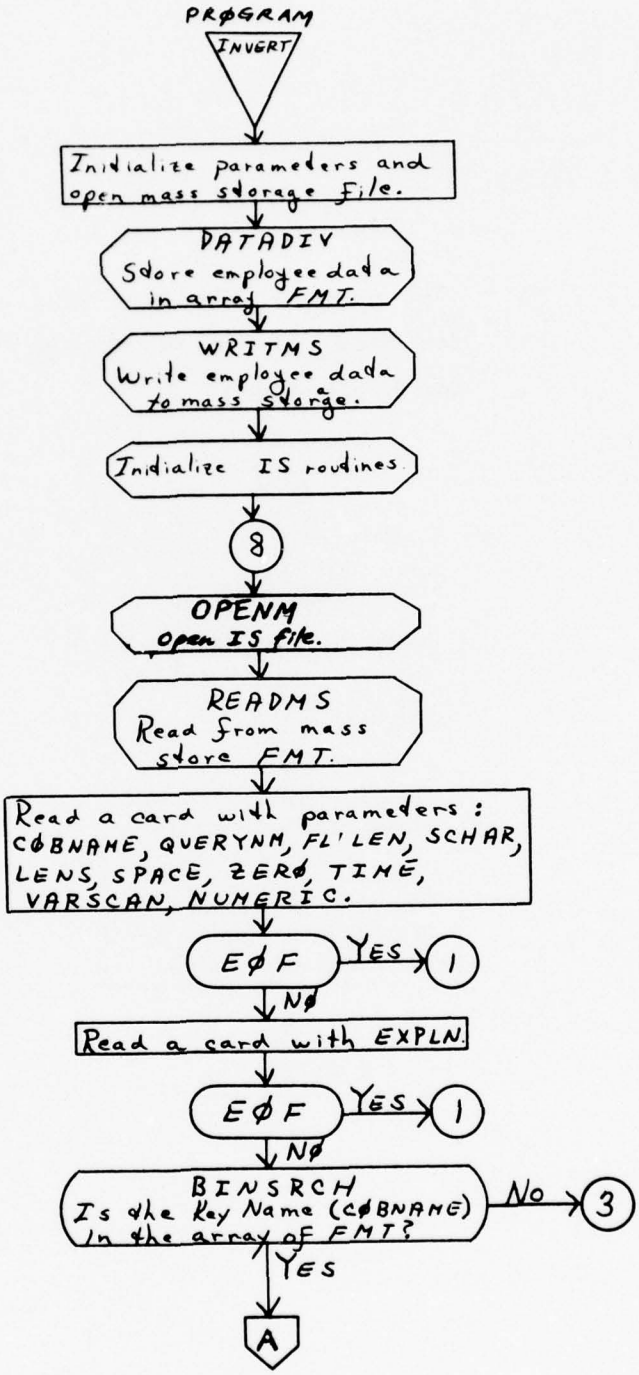
CATEG(NA) - the name of the category possessing the lesser number of header values.

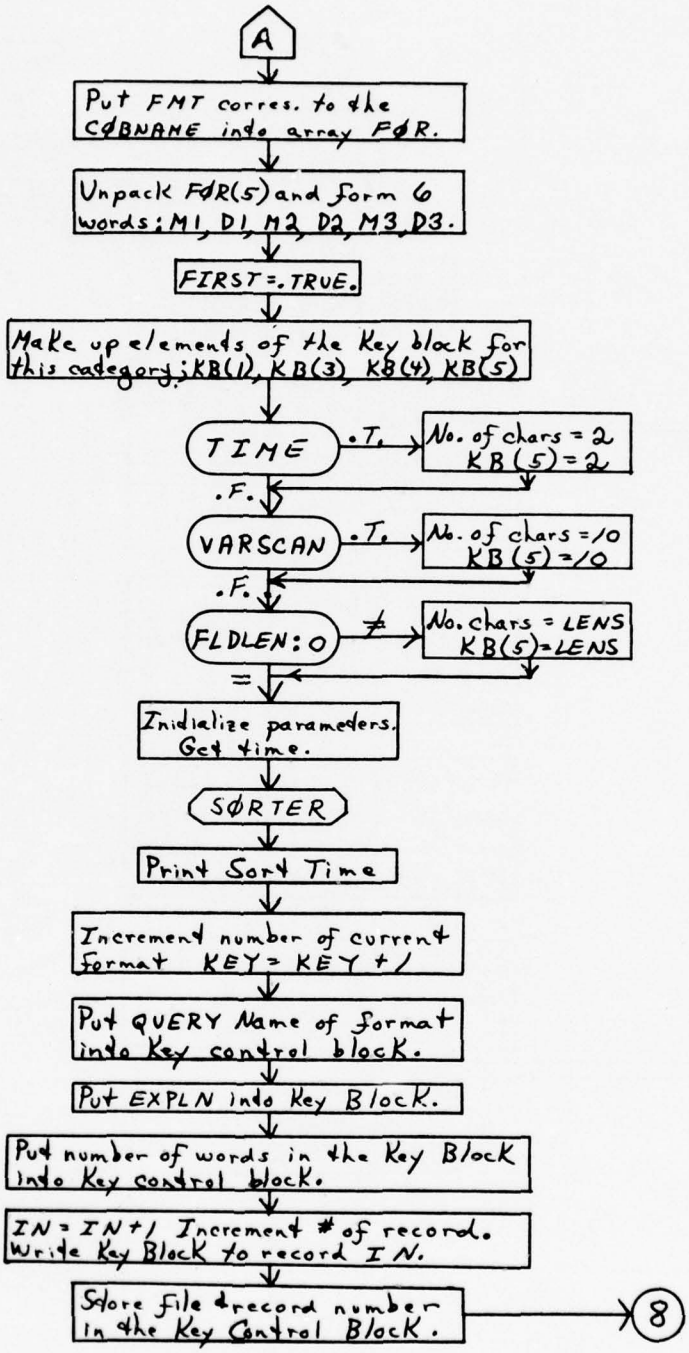
CATEG(NB) - the name of the category possessing the greater number of header values.

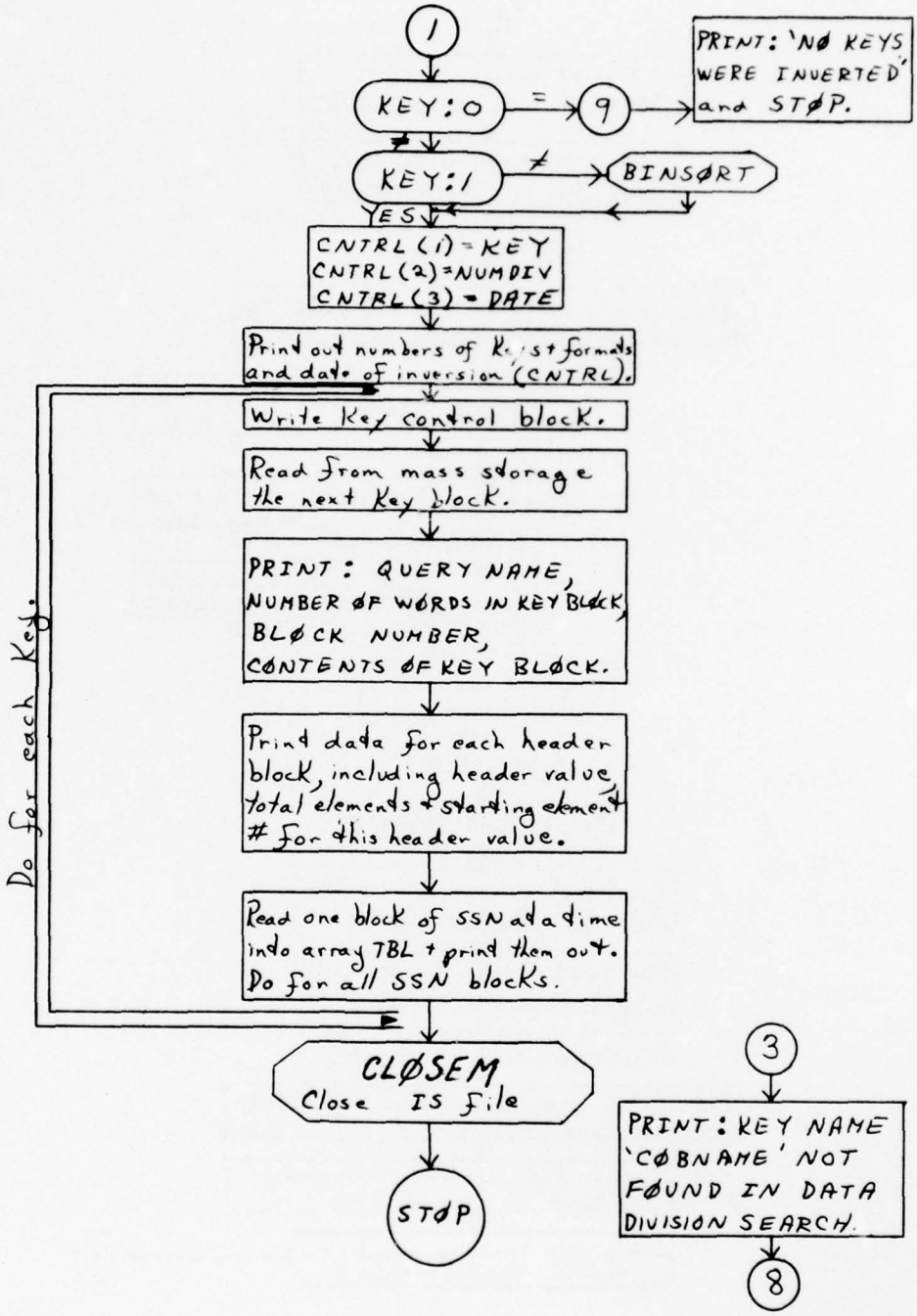
FILE(NA) - the name of the file which contains the data of the category with Key block KB(1, NA). (con't)

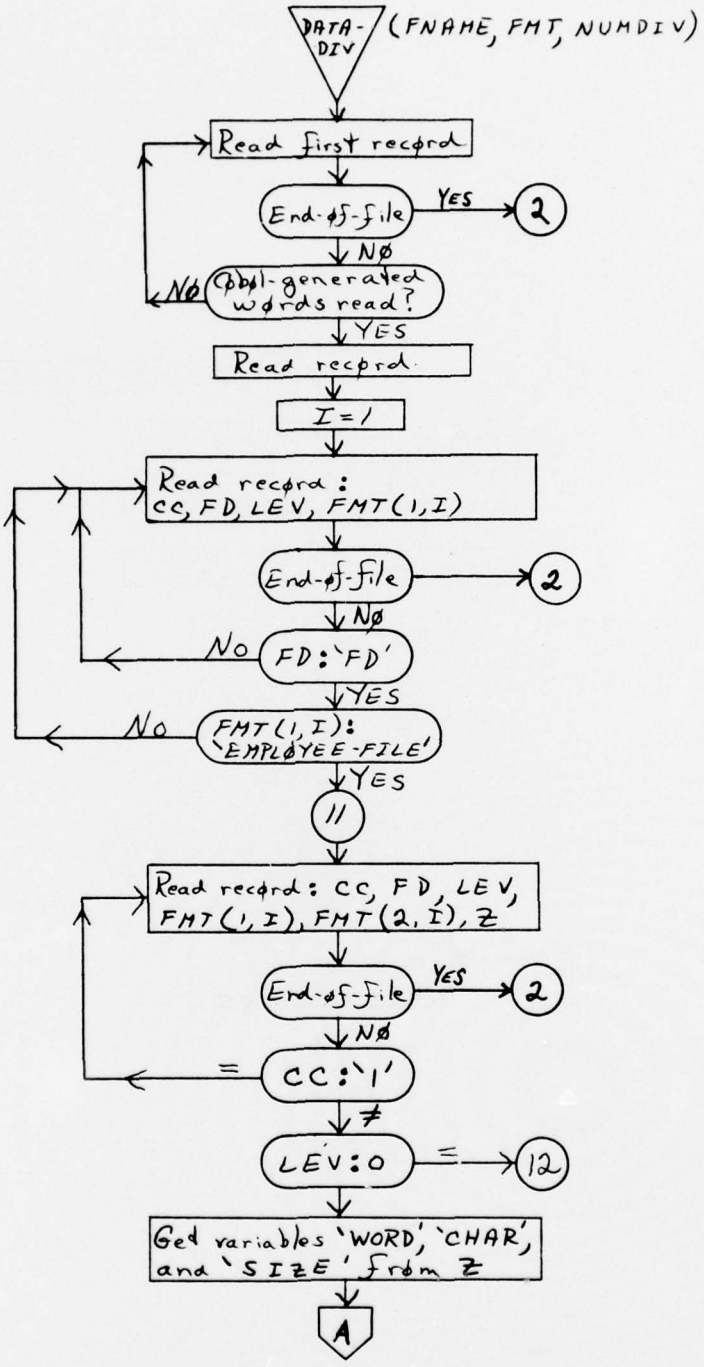
FILE (NB) – The name of the file which contains the data of the category with key block KB(1, NB)

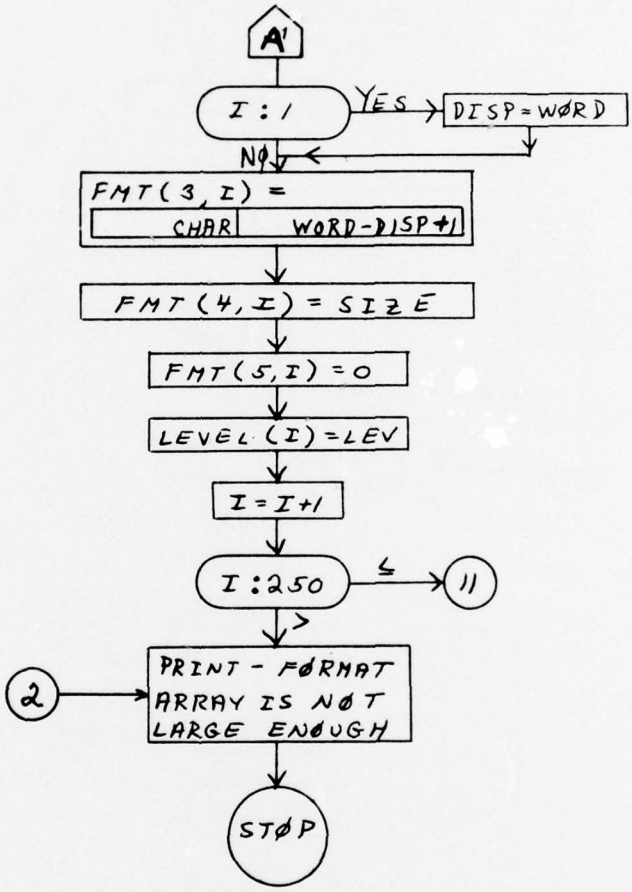


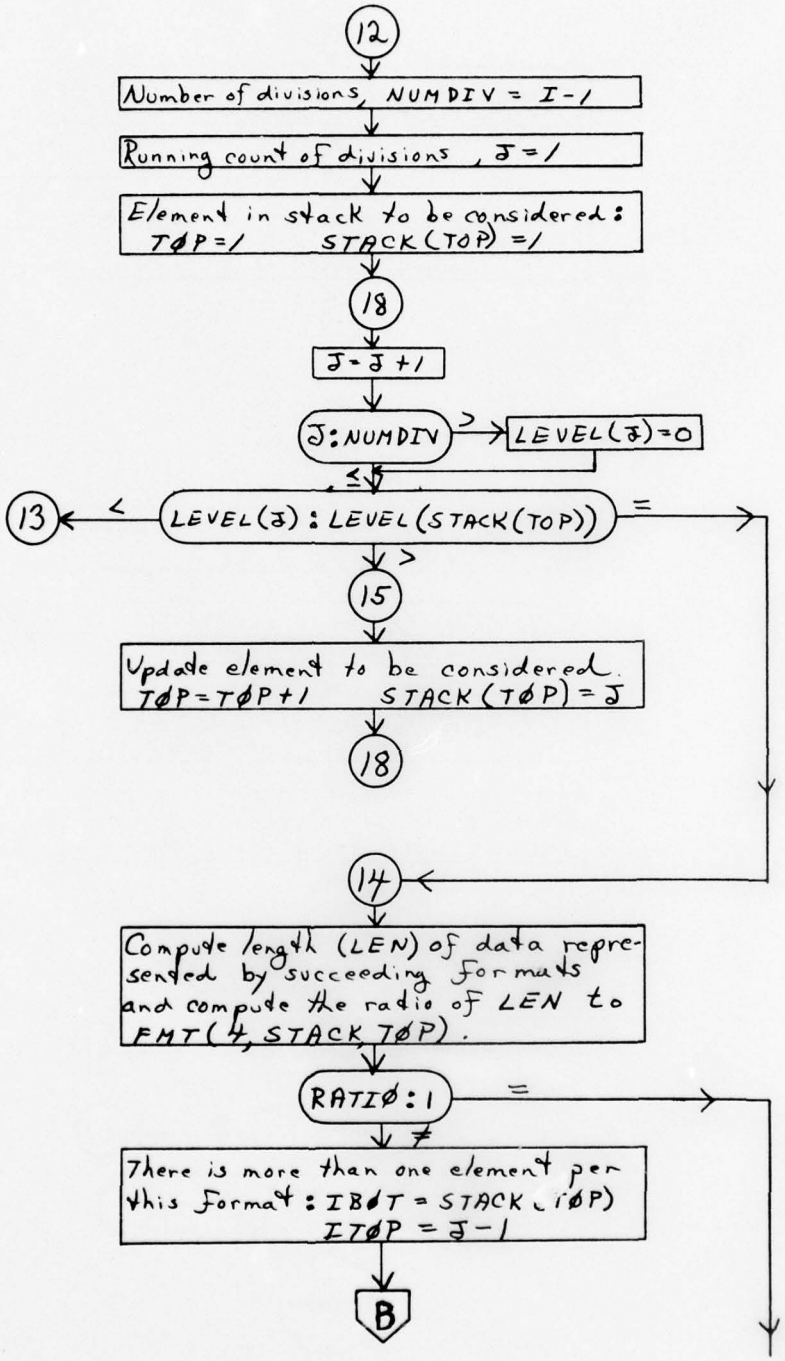

PROGRAM
INVERT
Initialize parameters and open mass storage file.
DATADIV
Store employee data in array FMT.
WRITMS
Write employee data to mass storage.
Initialize IS routines.
8
OPENM
Open IS file.
READMS
Read from mass store FMT.
Read a card with parameters: COBNAME, QUERYNM, FL'LEN, SCHAR, LENS, SPACE, ZERO, TIME, VARSCAN, NUMERIC.
EOF
YES
1
NO
Read a card with EXPLN.
EOF
YES
1
NO
BINSRCH
Is the Key Name (COBNAME) in the array of FMT?
No
3
YES
A

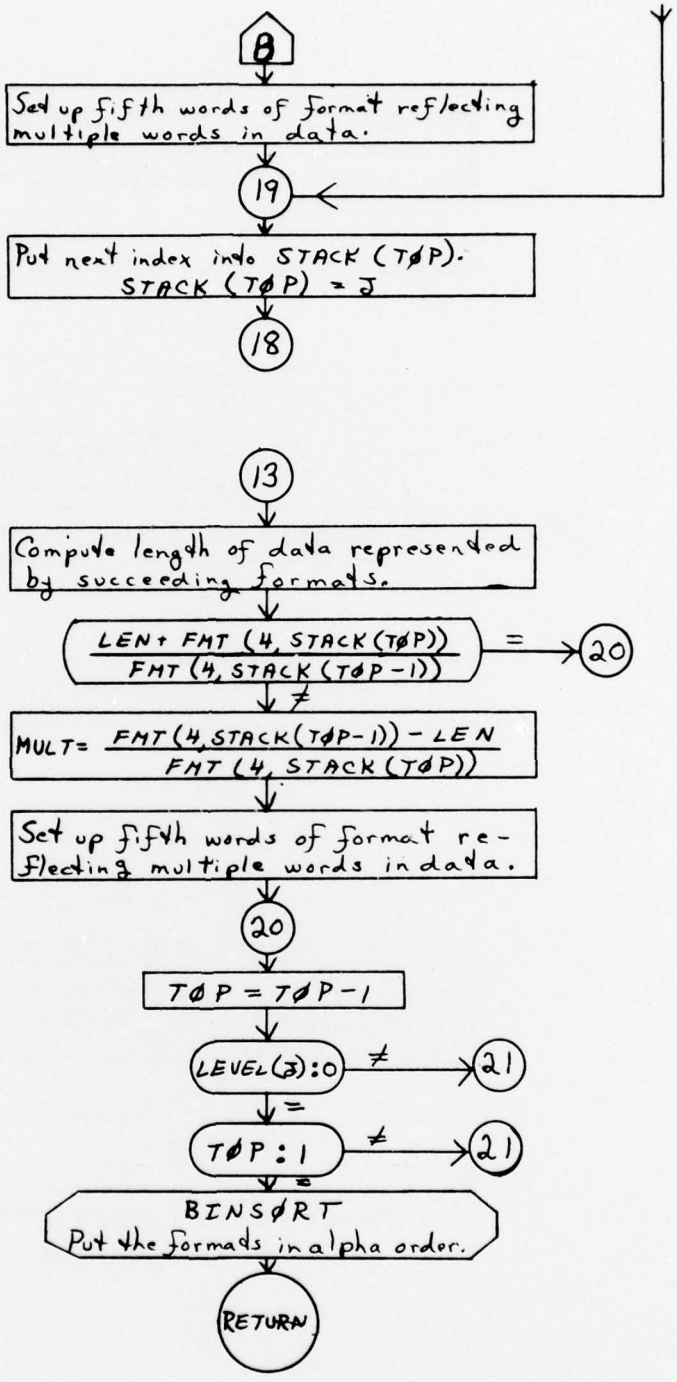

A
Put FMT corres. to the COBNAME into array FOR.
Unpack FOR(5) and form 6 words: M1, D1, M2, D2, M3, D3.
FIRST=.TRUE.
Make up elements of the Key block for this category; KB(1), KB(3), KB(4), KB(5)
TIME
.T.
No. of chars = 2
KB(5) = 2
.F.
VARSCAN
.T.
No. of chars = 10
KB(5) = 10
.F.
FLDLEN: 0
≠
No. chars = LENS
KB(5) = LENS
=
Initialize parameters. Get time.
SORTER
Print Sort Time
Increment number of current format KEY = KEY + 1
Put QUERY Name of format into Key control block.
Put EXPLN into Key Block.
Put number of words in the Key Block into Key control block.
IN = IN + 1 Increment # of record. Write Key Block to record IN.
Store File #record number in the Key Control Block.
8

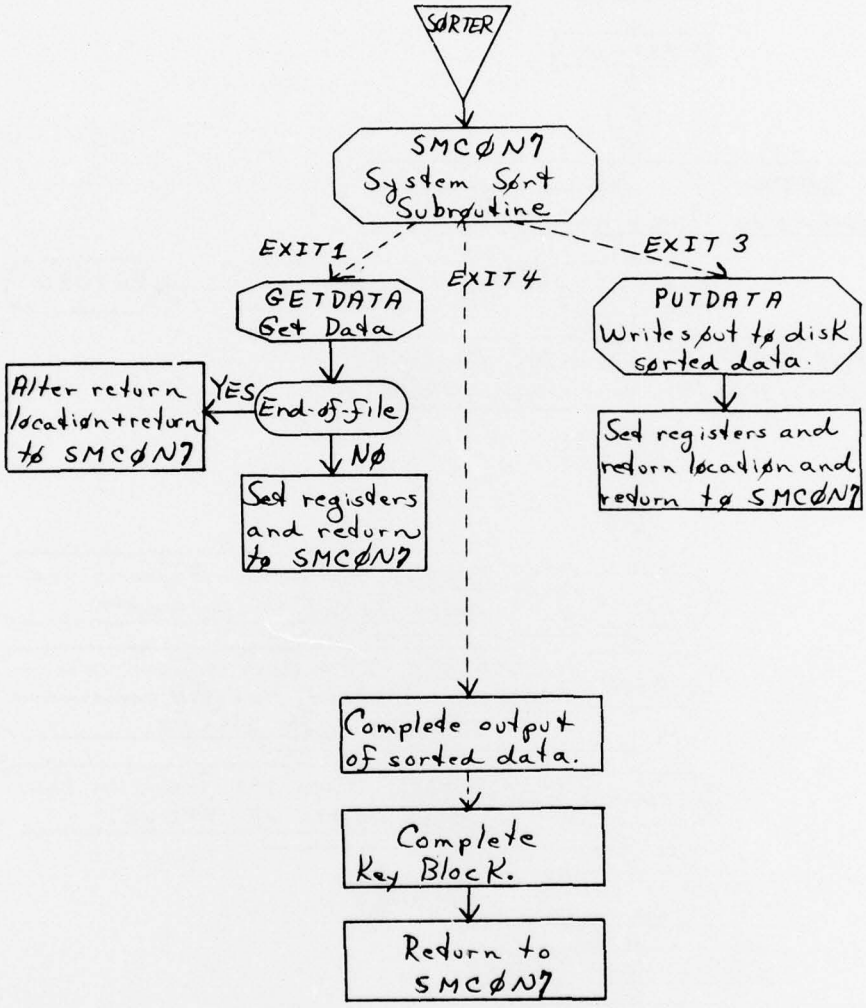

1
KEY:0
=
9
PRINT: 'NØ KEYS WERE INVERTED' and STØP.
≠
KEY:1
≠
BINSØRT
YES
CNTRL(1) = KEY
CNTRL(2) = NUMDIV
CNTRL(3) = DATE
Print out numbers of keys + formats and date of inversion (CNTRL).
Write key control block.
Read from mass storage the next key block.
PRINT: QUERY NAME, NUMBER ØF WØRDS IN KEY BLØCK, BLØCK NUMBER, CØNTENTS ØF KEY BLØCK.
Print data for each header block, including header value, total elements + starting element # for this header value.
Read one block of SSN at a time into array TBL + print them out. Do for all SSN blocks.
Do for each key.
CLØSEM
Close IS file
STØP
3
PRINT: KEY NAME 'CØBNAME' NOT FØUND IN DATA DIVISION SEARCH.
8

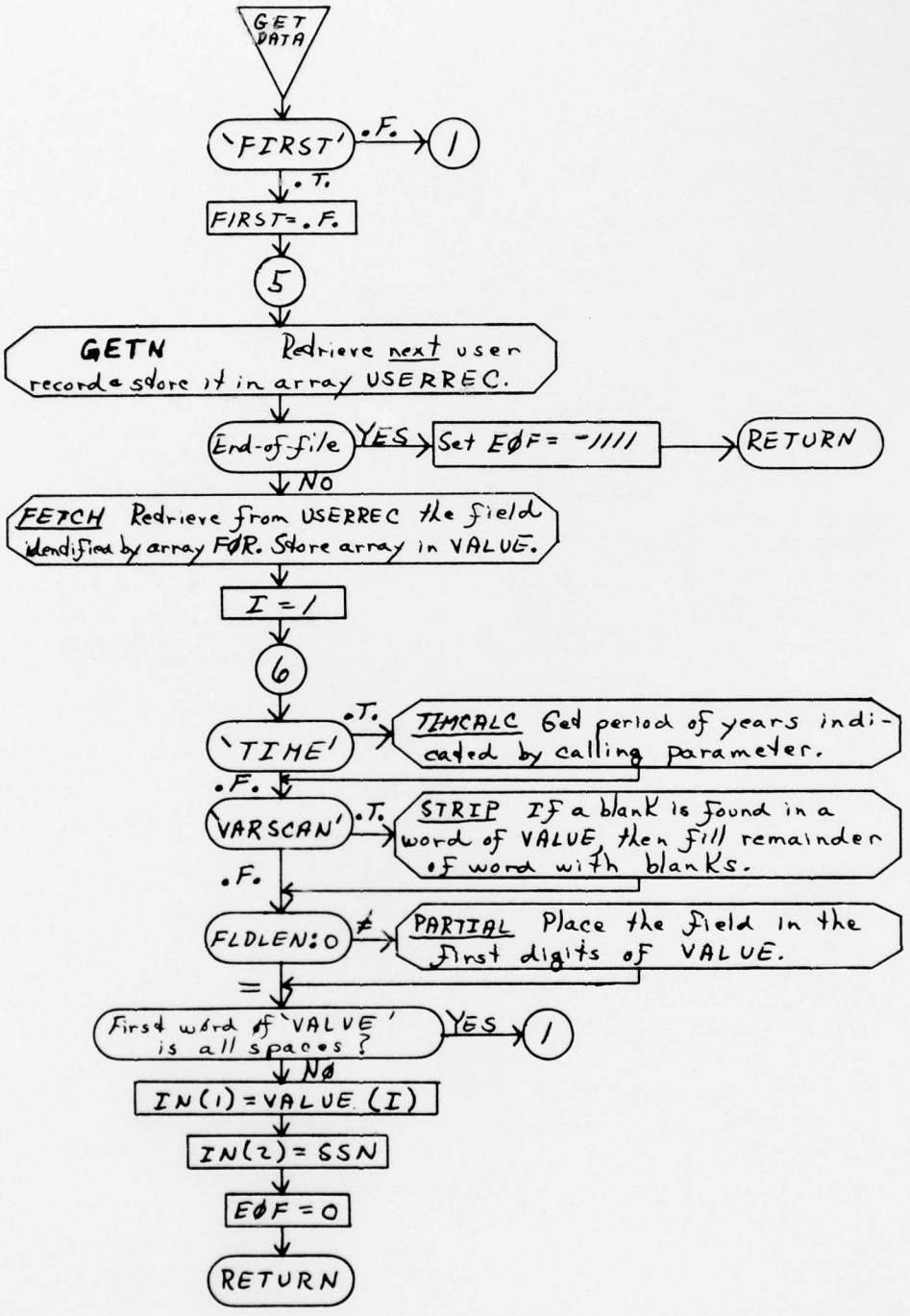

DATA-DIV
(FNAME, FMT, NUMDIV)
Read first record
End-of-file
YES
2
NO
Odd-generated words read?
NO
YES
Read record
I = 1
Read record: CC, FD, LEV, FMT(1,I)
End-of-file
2
NO
FD: 'FD'
NO
YES
FMT(1,I): 'EMPLOYEE-FILE'
NO
YES
11
Read record: CC, FD, LEV, FMT(1,I), FMT(2,I), Z
End-of-file
YES
2
NO
=
CC: '1'
≠
LEV: 0
=
12
Get variables 'WORD', 'CHAR', and 'SIZE' from Z
A

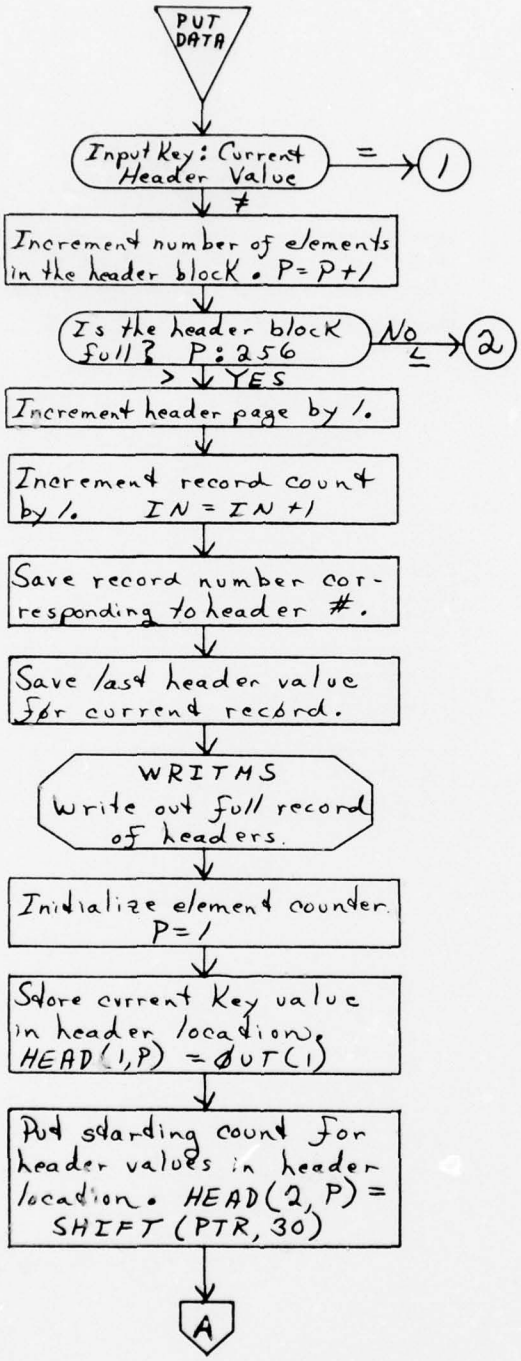

A'
I : 1
YES
DISP = WØRD
NØ
FMT(3, I) =
CHAR
WORD-DISP+1
FMT(4, I) = SIZE
FMT(5, I) = 0
LEVEL(I) = LEV
I = I+1
I : 250
≤
11
>
2
PRINT - FØRMAT
ARRAY IS NØT
LARGE ENØUGH
STØP

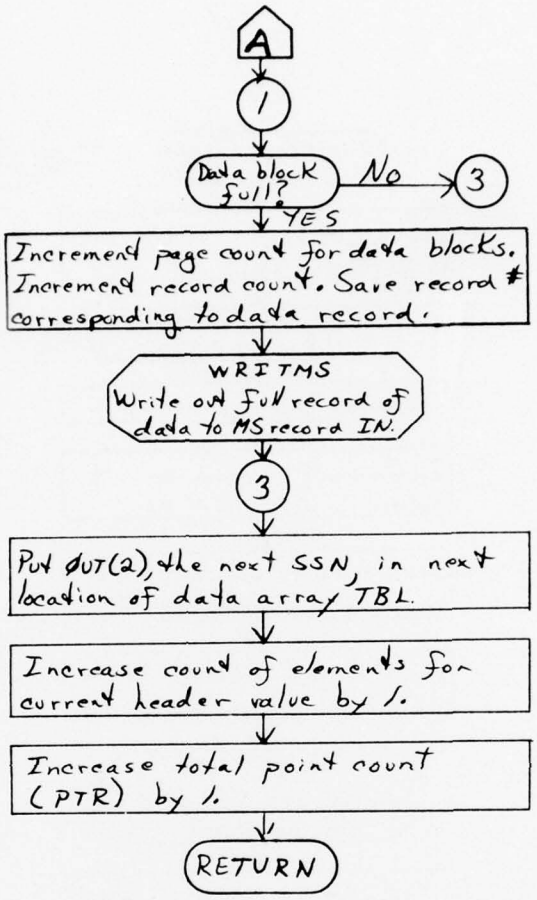

12
Number of divisions, NUMDIV = I-1
Running count of divisions, J = 1
Element in stack to be considered:
TOP = 1 STACK(TOP) = 1
18
J = J + 1
J : NUMDIV
>
LEVEL(J) = 0
≤
13
<
LEVEL(J) : LEVEL(STACK(TOP))
=
>
15
Update element to be considered.
TOP = TOP + 1 STACK(TOP) = J
18
14
Compute length (LEN) of data represented by succeeding formats and compute the ratio of LEN to FMT(4, STACK, TOP).
RATIO : 1
=
≠
There is more than one element per this format: IBOT = STACK(IOP)
ITOP = J - 1
B

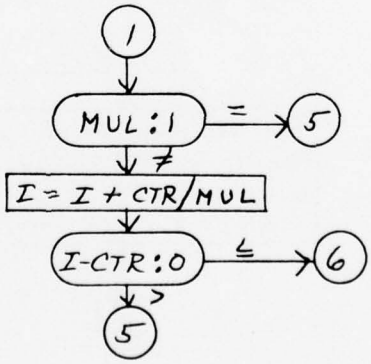

8
Set up fifth words of format reflecting multiple words in data.
19
Put next index into STACK (TØP).
STACK (TØP) = J
18
13
Compute length of data represented by succeeding formats.
LEN + FMT (4, STACK (TØP))
FMT (4, STACK (TØP-1))
=
20
≠
MULT = FMT (4, STACK (TØP-1)) - LEN
FMT (4, STACK (TØP))
Set up fifth words of format reflecting multiple words in data.
20
TØP = TØP-1
LEVEL(J):0
≠
21
=
TØP : 1
≠
21
=
BINSØRT
Put the formats in alpha order.
RETURN

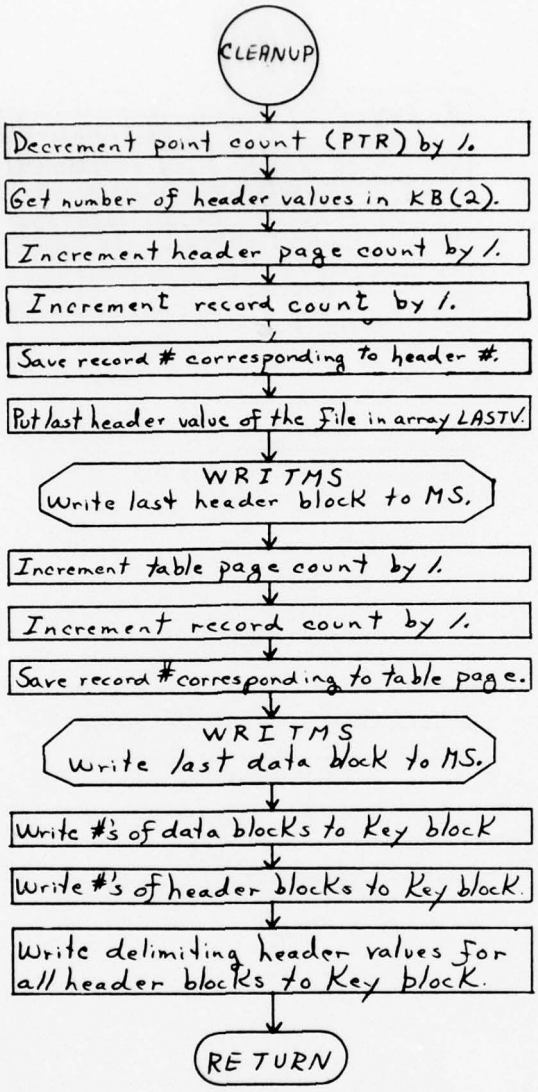

SORTER
SMCØN7
System Sort
Subroutine
EXIT1
EXIT 3
EXIT4
GETDATA
Get Data
PUTDATA
Writes out to disk
sorted data.
Alter return
location+return
to SMCØN7
YES
End-of-file
NØ
Set registers
and return
to SMCØN7
Set registers and
return location and
return to SMCØN7
Complede output
of sorted data.
Complete
Key Block.
Return to
SMCØN7

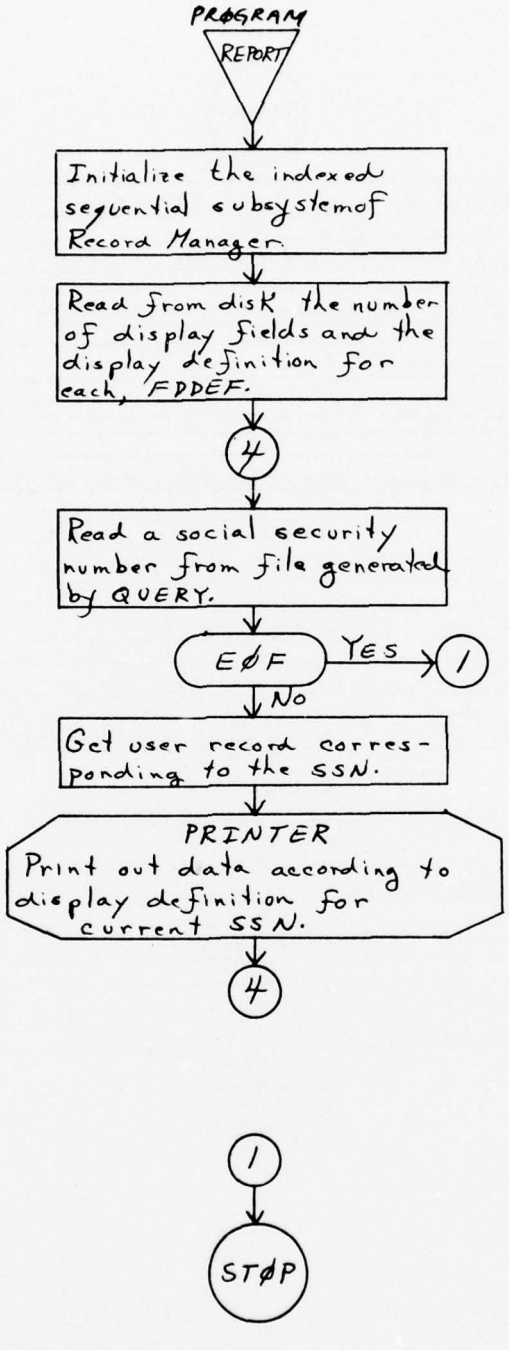

GET DATA
'FIRST'
.F.
1
.T.
FIRST=.F.
5
GETN Retrieve next user record & store it in array USERREC.
End-of-file
YES
Set EØF = -1111
RETURN
No
FETCH Retrieve from USERREC the field identified by array FØR. Store array in VALUE.
I=1
6
'TIME'
.T.
TIMCALC Get period of years indicated by calling parameter.
.F.
'VARSCAN'
.T.
STRIP If a blank is found in a word of VALUE, then fill remainder of word with blanks.
.F.
FLDLEN:0
≠
PARTIAL Place the field in the first digits of VALUE.
=
First word of 'VALUE' is all spaces?
YES
1
NØ
IN(1)=VALUE(I)
IN(2)=SSN
EØF=0
RETURN

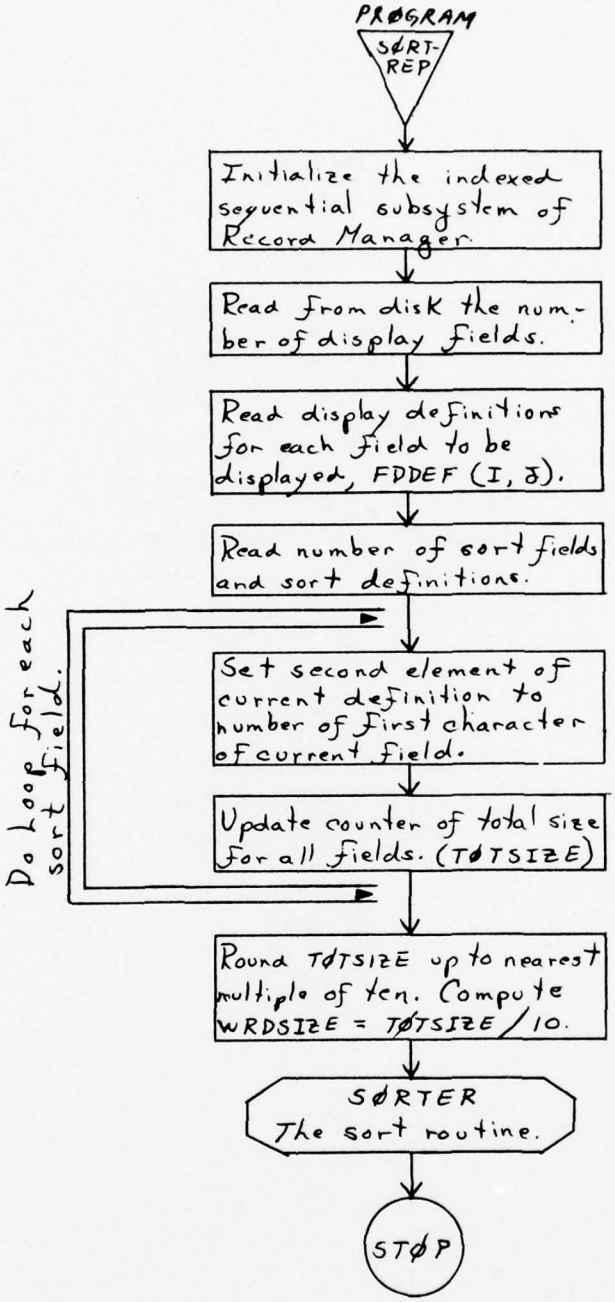

PUT DATA
Input Key: Current Header Value
=
1
≠
Increment number of elements in the header block. P=P+1
Is the header block full? P:256
No
2
>
YES
Increment header page by 1.
Increment record count by 1. IN = IN +1
Save record number corresponding to header #.
Save last header value for current record.
WRITMS
Write out full record of headers.
Initialize element counter. P=1
Store current Key value in header location. HEAD(1,P) = OUT(1)
Put starting count for header values in header location. HEAD(2, P) = SHIFT (PTR, 30)
A

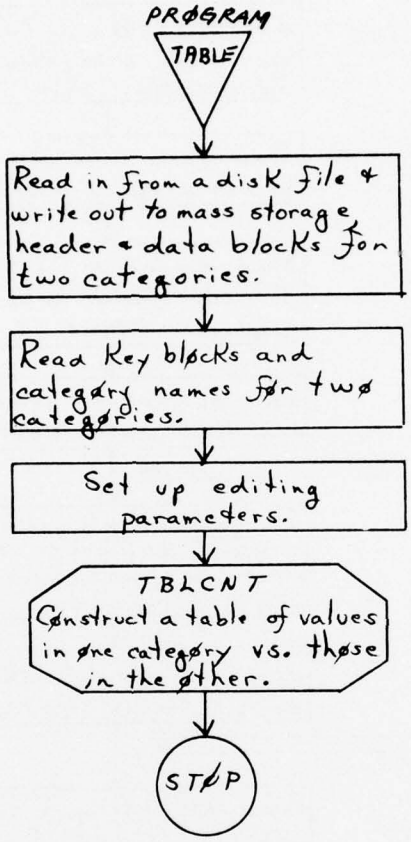

A
1
Data block full?
No
3
YES
Increment page count for data blocks. Increment record count. Save record # corresponding to data record.
WRITMS
Write out full record of data to MS record IN.
3
Put OUT(2), the next SSN, in next location of data array TBL.
Increase count of elements for current header value by 1.
Increase total point count (PTR) by 1.
RETURN


1
MUL:1
=
5
≠
I = I + CTR/MUL
I-CTR:0
≤
6
>
5


CLEANUP
Decrement point count (PTR) by 1.
Get number of header values in KB(2).
Increment header page count by 1.
Increment record count by 1.
Save record # corresponding to header #.
Put last header value of the file in array LASTV.
WRITMS
Write last header block to MS.
Increment table page count by 1.
Increment record count by 1.
Save record # corresponding to table page.
WRITMS
Write last data block to MS.
Write #'s of data blocks to Key block
Write #'s of header blocks to Key block.
Write delimiting header values for all header blocks to Key block.
RETURN


PROGRAM
REPORT
Initialize the indexed sequential subsystem of Record Manager.
Read from disk the number of display fields and the display definition for each, FDDEF.
4
Read a social security number from file generated by QUERY.
EOF
YES
1
No
Get user record corresponding to the SSN.
PRINTER
Print out data according to display definition for current SSN.
4
1
STOP


PROGRAM
SØRT-REP
Initialize the indexed sequential subsystem of Record Manager.
Read from disk the number of display fields.
Read display definitions for each field to be displayed, FDDEF (I, 8).
Read number of sort fields and sort definitions.
Do Loop for each sort field.
Set second element of current definition to number of first character of current field.
Update counter of total size for all fields. (TØTSIZE)
Round TØTSIZE up to nearest multiple of ten. Compute WRDSIZE = TØTSIZE / 10.
SØRTER
The sort routine.
STØP


PROGRAM
TABLE
Read in from a disk file & write out to mass storage, header & data blocks for two categories.
Read Key blocks and category names for two categories.
Set up editing parameters.
TBLCNT
Construct a table of values in one category vs. those in the other.
STOP

INITIAL DISTRIBUTION

Copies	
12	DDC

CENTER DISTRIBUTION

Copies	Code	
1	18/1808	G.H. Gleissner
1	1805	E. Cuthill
2	1809	D. Harris
1	182	A. Camara
5	182	P. Battey
1	1844	S. Wybraniec
1	189	G. Gray
50	712.9	B. Pierce
30	5214.1	Reports Distribution
1	522.1	Library (C)
1	522.2	Library (A)

DTNSRDC ISSUES THREE TYPES OF REPORTS

(1) DTNSRDC REPORTS, A FORMAL SERIES PUBLISHING INFORMATION OF PERMANENT TECHNICAL VALUE, DESIGNATED BY A SERIAL REPORT NUMBER.

(2) DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, RECORDING INFORMATION OF A PRELIMINARY OR TEMPORARY NATURE, OR OF LIMITED INTEREST OR SIGNIFICANCE, CARRYING A DEPARTMENTAL ALPHANUMERIC IDENTIFICATION.

(3) TECHNICAL MEMORANDA, AN INFORMAL SERIES, USUALLY INTERNAL WORKING PAPERS OR DIRECT REPORTS TO SPONSORS, NUMBERED AS TM SERIES REPORTS; NOT FOR GENERAL DISTRIBUTION.